Generating OpenAPI Specifications from Online API Documentation with Large Language Models

Koren Lazar[*] and **Matan Vetzler** and **Kiran Kate** and **Jason Tsay** and **David Boaz**
Himanshu Gupta and **Avraham Shinnar** and **Rohith D Vallam** and **David Amid**
Esther Goldbraich and **Guy Uziel** and **Jim Laredo** and **Ateret Anaby Tavor**
IBM Research

Abstract

AI agents and business automation tools interacting with external web services require standardized, machine-readable information about their APIs in the form of API specifications. However, the information about APIs available online is often presented as unstructured, free-form HTML documentation, requiring external users to spend significant time manually converting it into a structured format. To address this, we introduce OASBuilder, a novel framework that transforms long and diverse API documentation pages into consistent, machine-readable API specifications. This is achieved through a carefully crafted pipeline that integrates large language models and rule-based algorithms which are guided by domain knowledge of the structure of documentation webpages. Our experiments demonstrate that OASBuilder generalizes well across hundreds of APIs, and produces valid OpenAPI specifications that encapsulate most of the information from the original documentation. OASBuilder has been successfully implemented in an enterprise environment, saving thousands of hours of manual effort and making hundreds of complex enterprise APIs accessible as tools for LLMs.

1 Introduction

AI agents have gained significant popularity in automating tasks across diverse domains, from finance to customer service (George and George, 2023), complementing traditional rule-based business automation systems. Both AI agents and automation systems depend on various external APIs to function effectively. For AI agents, APIs serve as tools to access external resources such as real-time data and integration with external services, while for automation systems, APIs are integral to building automated workflows. However, efficient interaction with these APIs requires that their information be provided in a standardized, machine-readable format. The OpenAPI Specification (OAS)[1] is the leading format for documenting REST APIs (Espinoza-Arias et al., 2020), providing a structured, compact representation compatible with large language model (LLM) frameworks such as LangChain (Chase, 2022).

Unfortunately, many API providers do not provide standardized API specifications. Our analysis of the 14 most popular APIs on Postman for 2023[2] revealed that only five providers publicly share their OAS (see Appendix A.6 for details). Instead, most offer online API documentation presented as HTML webpages with human-readable hypertext describing the API operations. These webpages frequently lack structural consistency and fail to follow standard conventions (Danielsen and Jeffrey, 2013). As a result, developers often need to manually convert documentation into OAS format, a labor-intensive error-prone task, especially for real-world APIs, which are typically large and complex. This challenge has sparked the search for automated solutions to convert API documentation webpages into OAS documents (Cao et al., 2017; Yang et al., 2018; Bahrami and Chen, 2020; Andročec and Tomašić, 2023). However, existing approaches, whether based on automatic parsing or the direct application of LLMs, have consistently struggled to produce accurate and complete OAS documents. These challenges stem from significant variability in API documentation formats, inconsistencies in layout, the presence of embedded JavaScript-generated content, and the considerable length of documentation webpages, often amounting to millions of words.

To address these challenges, we introduce OASBuilder, an innovative LLM-based end-to-end framework for automating the generation of OAS from API documentation webpages. OASBuilder

[*]Corresponding author: `koren.lazar@ibm.com`

[1]https://swagger.io/specification/

[2]`https://www.postman.com/explore/most-popular-apis-this-year`

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track), pages 237–253
July 28-30, 2025 ©2025 Association for Computational Linguistics

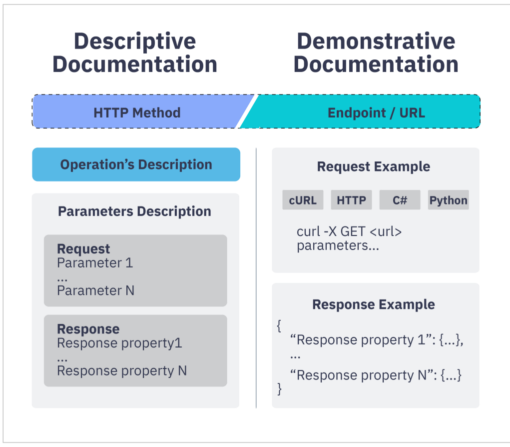

Figure 1: Typical structure of API operation documentation on webpages, featuring the descriptive documentation on the right, the demonstrative documentation on the left, and the operation signature at the top, which can appear on either side. A real-world example from Shopify API can be found in Appendix A.1.

employs a multi-stage approach, breaking the OAS generation process into smaller, manageable subtasks. It scrapes API documentation pages, segments them into sections corresponding to individual API operations, and filters out irrelevant content. Then, the documentation for each API operation (analogous to a function) is then translated into OAS via parallel LLM calls. Finally, OASBuilder provides an intuitive platform for manual validation and editing, supported by evidence from the source webpage and AI-based tools for refining the OAS, ensuring a reliable, high-quality result while significantly reducing manual effort.

To the best of our knowledge, OASBuilder is the *first LLM-based automated system* for generating OAS from API documentation pages. Our empirical experiments highlight its ability to handle diverse API documentation formats, producing accurate and comprehensive OAS documents. Furthermore, OASBuilder has been successfully deployed in an enterprise environment, where it has generated hundreds of API specifications, saving developers thousands of hours of work.[3]

2 OpenAPIs and Documentation Websites

The OpenAPI Specification (OAS) is a standardized framework for formally describing RESTful APIs, specifying their operations (analogous to functions or tools), authentication mechanisms, and other operational details. In enterprise applications, OAS documents are typically extensive, comprising numerous operations and deeply nested request and response objects, and frequently span thousands of lines. A minimal example is presented on the right side of Figure 2.

OAS is especially valuable for AI agents as it offers a concise and standardized representation of each API. This allows LLMs to dynamically understand and interact with multiple APIs—a crucial ability for autonomous agents that must reason about and utilize external services. Moreover, OAS facilitates automation in development processes, such as generating client libraries, server stubs, and documentation, streamlining workflows, reducing manual effort, and minimizing errors.

Despite these benefits, many API service providers only offer online documentation in HTML format, which often lacks consistency in structure or adherence to any standard conventions. Although this documentation does not adhere to any convention which makes automatic parsing infeasible, document pages often contain recurring semantic components which complements each other. As shown in Figure 1, these components generally consist of an **operation signature**, specifying the HTTP method and path (e.g., `GET /status`); **descriptive documentation**, providing a textual overview of the operation's purpose, security details, and a tabular breakdown of request and response fields, including field names, data types, formats, required/optional status, and descriptions; and **demonstrative documentation**, featuring usage examples such as a sample request (e.g., a cURL command) and a sample response (e.g., a JSON object). For a real-world example of such documentation, see Appendix A.1.

Although OASBuilder leverages each of these components to generate a more comprehensive specification, its sole assumption is the presence of an operation signature or a request example to identify the operations, as detailed in Section 3.1.

3 OASBuilder

OASBuilder consists of three main components: (1) an automated method for generating an OAS from a webpage, described in Sections 3.1, 3.2, and 3.3; (2) AI-powered enhancement of the OAS, detailed in Section 3.4; and (3) a user-friendly in-

[3] `https://www.ibm.com/docs/en/watsonx/watson-orchestrate/current?topic=skills-using-openapi-builder`.

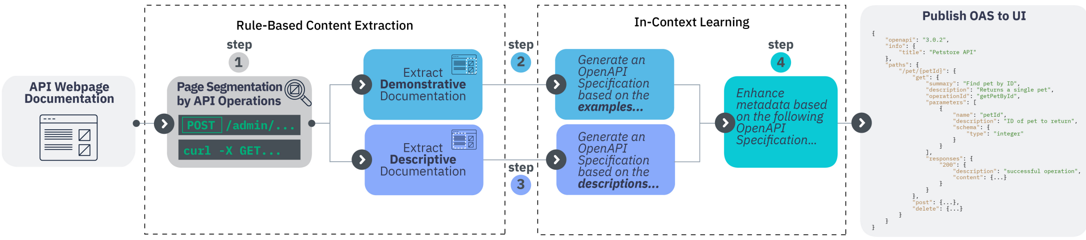

Figure 2: OASBuilder pipeline: The system first segments the webpage based on identified API operations. Then, for each operation, it searches for demonstrative and descriptive documentation. The documentations are then processed by an LLM using in-context learning to generate two partial OAS, which are merged into a complete, grounded OAS. The OAS is subsequently enhanced to fill in missing metadata without external resources. Finally, the OAS is presented to the user for final revisions.

terface for viewing, editing, and validating OAS documents, which also integrates the AI-powered enhancement to further accelerate the final revisions.

Given the potential length of documentation and LLM input limits, the pipeline adopts a modular approach. The generation task is divided into smaller sub-tasks, whose outputs are then combined to create a cohesive result.

Throughout the pipeline, LLM generation was based on in-context learning, as fine-tuning was not feasible due to the lack of labeled data. It is assumed that the LLM was exposed to OAS during its pretraining. Figure 2 illustrates the pipeline.

3.1 Scraping

The first step is scraping the API documentation webpage, with the goal of segmenting it into continuous sections, each linked to a specific operation through the identification of operation signatures and usage examples. The application begins by launching a web browser and navigating to the specified URL. To achieve this, it utilizes Selenium,[4] a tool that provides capabilities for controlling a web browser, enabling us to interact with and query the webpage HTML elements. Since many pages load part of their content dynamically through user interactions, the system uses a small set of rules to detect and click on elements like "expand all" or "example". After the content is loaded, OASBuilder identifies operation signatures and API request examples. Specifically, it searches for HTML elements whose text includes cURL commands, HTTP commands, or patterns resembling operation signatures, such as <HTTP_METHOD ENDPOINT>. OASBuilder assumes that all instances of a specific operation appear sequentially on the page. Hence, it defines the boundaries of each operation as the section spanning from the first title preceding its initial instance to the first title marking the start of the next operation.

3.2 Demonstrative OAS Generation

After segmenting the webpage into operations, as outlined in Section 3.1, the next step (step 2 in Figure 2) involves extracting demonstrative examples and generating an OAS from them. Since the system has already extracted request examples in the previous stage, it only needs to identify the corresponding request bodies and response examples within the operation boundaries, if available.

After extracting demonstrative examples, the system converts them into OAS format by decomposing the process into multiple parallel LLM calls. We favor LLMs over complex rule-based parsing, as the latter is prone to errors arising from human mistakes in example creation and noise introduced during automated scraping. This multi-stage approach addresses input length limitations while enabling efficient parallel generation. First, OASBuilder generates a partial OAS for each request example, excluding the request body. To do so, each example is standardized into a canonical cURL command to minimize variations. Parallel LLM calls are then employed, each using two diverse in-context examples drawn from real-world scenarios to convert the standardized commands into partial OAS documents. These include essential metadata such as servers, paths, HTTP methods, and request parameters. Second, the system generates JSON schemas for both request bodies and

[4] https://www.selenium.dev/

response examples. Due to the difficulty LLMs face in processing large, deeply nested JSON structures—a well-documented issue (Shorten et al., 2024)—OASBuilder divides these structures into smaller fragments based on a predefined line threshold, preserving scope boundaries. Parallel LLM calls are then applied to each fragment, using two in-context examples per call, to generate the corresponding JSON schemas. Prompt examples are provided in Appendix A.3.

3.3 Descriptive OAS Generation

In parallel with generating an OAS from the demonstrative examples discussed in Section 3.2, OASBuilder also generates a second OAS based on the descriptive documentation (see Section 3.2 for more details). This stage is labeled as step 3 in Figure 2.

Parsing these documentations using deterministic rule-based algorithms is impractical due to the significant variations in HTML structures across websites (Yang et al., 2018). Therefore, leveraging the capabilities of contemporary LLMs offers a more viable solution, as they can generalize over such structural differences effectively.

Therefore, OASBuilder first applies a search algorithm to extract the descriptive documentation from the operation's scope identified in the scraping stage. The algorithm employs various heuristics to identify and filter the appropriate HTML elements, leveraging prior knowledge of the high-level structure of these webpages. For instance, for a webpage containing a single operation, it identifies the smallest HTML scope that includes both an operation signature or a request example, while maximizing the number of HTML elements that their text is equal to a parameter name from the example. Additionally, usage examples and any HTML elements lacking indicators of relevant information are excluded. For more details, see Appendix A.2 and algorithm 1.

After narrowing the scope to a relatively small number of HTML elements, the LLM input is further reduced by filtering out the HTML attributes (e.g., css styles) as they often do not contain any relevant information. An LLM is then employed using in-context learning to convert the extracted HTML into an OAS. We found that LLMs often require exposure to various structures in the examples to correctly apply them in the test case. For example, the model would fail to generate a requestBody component or an enum attribute if they were not provided in the input example. Therefore, the in-context example was carefully crafted using data from multiple real-world APIs with diverse structures and attributes. In the case of context window overflow, it retries with an alternative shorter in-context example. A prompt example is provided in Appendix A.3. After generating the OAS, the system validates its structure and verify that all the generated parameter names appear in the input to prevent hallucinations.

Lastly, the generated OAS is merged with the one created in Section 3.2 to yield a final comprehensive OAS. In this integration, the description and required attributes from the descriptive documentation are prioritized, while the type and location fields from the demonstrative documentation take precedence. This prioritization is determined based on the reliability of the attribute in each type of documentation.

3.4 OAS Enhancement

After generating an OAS from the documentation, OASBuilder enriches missing metadata using AI-based tools based on information within the OAS (step 4 in Figure 2). These tools perform two key functions: (1) extracting parameter metadata from grounded parameter descriptions and (2) generating missing metadata based on its surrounding context. This enriched metadata, including descriptions, enums, and defaults, is essential for accurately documenting API behavior and supports various downstream tasks, such as conversational agents, slot filling (Vaziri et al., 2017), test generation (Kim et al., 2022), and API sequencing.

Metadata extraction from parameter descriptions: Parameter descriptions often include metadata like enum, default, and format, which can be explicitly defined in the OAS. LLMs are well-suited for extracting this metadata. To improve extraction, we designed prompts using in-context examples that handle both explicit (e.g., "the default value is 10") and implicit metadata (e.g., "the option is disabled by default"). To avoid hallucinations, the extracted values are verified to match the descriptions. To minimize LLM calls, OASBuilder used two strategies: First, a keyword-based filtering mechanism that triggers LLM calls only when relevant terms like "default" or "not provided" are present, saving over 90% of calls for many metadata fields. Second, the prompts were designed to process multiple descriptions at once, further reducing LLM calls.

Metadata Generation Using OAS Structure: OASBuilder addresses missing method and parameter descriptions, as well as parameter examples, in an OAS by utilizing LLM-based prompts to generate the missing metadata. Relevant context from the OAS is extracted and provided as input to these prompts. For example, the context for generating parameter descriptions and examples are the parameter name, the method name and description and the parameter description for the example generation. For method descriptions, it incorporates the method name, endpoint path, and operation ID. Examples of the generated metadata is provided in Appendix A.5.

4 Experiments

This section presents the results of a series of experiments conducted to evaluate the performance of OASBuilder. We utilized several well-known open-source LLMs with in-context prompting capabilities, including llama-3-70b-instruct (Touvron et al., 2023), codellama-34b-instruct (Rozière et al., 2024), mistral-7b-instruct (Jiang et al., 2023), mixtral-8x7b-instruct (Jiang et al., 2024), and granite-20b-code-instruct (Mishra et al., 2024). The prompts were not fine-tuned for any specific model. Baselines were not included, as previous studies neither evaluated on a public benchmark nor provided their code or reproduction details.

4.1 Syntactic Evaluation

In this section, we analyze the syntactic properties of OAS documents generated by OASBuilder using various LLMs on a corpus of 50 diverse documentation webpages covering 189 operations (see Appendix A.4 for details).

The evaluation focuses on three metrics: (1) the proportion of outputs that are valid JSONs; (2) the proportion that qualify as valid OAS documents; and (3) the average number of errors in valid JSONs. While the latter two metrics are related—errors occur only in invalid OAS documents—quantifying the errors provides insight into the degree of syntactic deviation, helping to estimate the effort required for correction. We computed the two metrics with jsonschema library.[5]

Table 1 presents the results of the syntactic analysis. Notably, granite-code and codellama emerge as

	VALID JSON	VALID OAS	ERRORS
CODELLAMA	.99	**.89**	.59
GRANITE-CODE	**1**	.73	**.48**
LLAMA-3	**1**	.29	.78
MISTRAL	**1**	.4	.54
MIXTRAL	.92	.66	.64

Table 1: Syntactic evaluation results for OAS generation by different LLMs on 50 web pages covering 189 operations. Metrics include the ratios of valid JSONs and OAS documents and the average errors in valid JSONs.

the top-performing models. This likely reflects the prevalence of JSON-related tasks in code-oriented benchmarks. While codellama achieved the highest proportion of valid OAS, its error rate ranked third among the models. In contrast, granite-code produced the second-highest rate of valid OAS while exhibiting the lowest error incidence. The remaining models generally succeeded in generating valid JSON but showed considerably lower and more variable rates of valid OAS generation. These findings suggest that, even when decomposed into subtasks, OAS generation remains a nontrivial challenge for LLMs.

To evaluate scalability, we collected a larger dataset of 291 API documentation URLs and repeated the experiment using the granite-code model. Results showed that 100% of the outputs were valid JSON, 89% were valid OAS, and the average number of errors per OAS was 0.17. Furthermore, 86% of the OAS documents contained at least one operation and one parameter.

Lastly, we attempted to generate OAS documents using GPT-4-128K (OpenAI et al., 2024) directly from the original HTML, without using OASBuilder. We found that the model was able to generate a valid OAS only for 25% of the webpages. This outcome is not unexpected, as many of these webpages contain much more than 128K tokens, the model's context window limit.

4.2 Semantic Evaluation

In this section, we assess the overall capabilities of OASBuilder to generate rich and complete OAS given various API documentation webpages. To that end, we employed a manually labeled dataset comprising of 108 operations containing thousands of parameters and properties from different API documentation websites. We conducted experiments to compare the enhanced OAS documents generated by OASBuilder with the ground truth

[5]https://github.com/python-jsonschema/jsonschema

	REQUEST							RESPONSE			
MODEL	P	R	F1	DESC.	REQ.	DEF.	ENUM	P	R	F1	DESC.
CODELLAMA	.95	.85	.90	.89	**.88**	.88	.69	.93	.56	.70	.68
GRANITE-CODE	**.96**	**.86**	**.91**	.90	**.88**	.84	.76	.92	.54	.68	.79
LLAMA-3	**.96**	.78	.86	**.91**	.87	.92	**.81**	**.97**	**.62**	**.75**	.72
MISTRAL	.94	.67	.78	.75	.85	.56	.50	.90	.57	.69	.64
MIXTRAL	.95	.55	.70	.87	.80	**.93**	.62	.90	.52	.65	**.84**

Table 2: End-to-end results of OASBuilder for different LLMs on 108 different operations containing thousands of parameters and properties. We report the precision (P), recall (R), F1-score (F1) of the parameters, and the cosine similarity of the descriptions, as well as the F1-score of required (Req.), default (Def.) and enum attributes. All results are averaged across all parameters and were based on the valid OASs for each model.

OAS, using different LLMs. In all experiments we used the in-context learning approach with the same prompt and in-context examples across models.

Table 2 presents the end-to-end results. First, the parameter precision for all models was relatively high, suggesting that hallucinations were uncommon. The recall for request parameters was also high, particularly for granite-code and codellama, with values of 0.86 and 0.85, respectively. Additionally, the description similarity and the F1 scores for the required, default, and enum attributes were relatively high. These findings indicate that, although the generated OAS documents are not perfect, they capture most of the relevant information on the request side, significantly reducing the user's manual annotation effort. Since many request parameters and their attributes such as default, enum, and description are found exclusively in the descriptive documentation, we can conclude that the information from the descriptive documentation were successfully integrated in the final OAS.

Lastly, the recall for response generation was lower, likely due to the highly nested and lengthy structure of many responses, as well as the frequent lack of descriptive documentation for response properties. Overall, the LLMs demonstrated competitive performance, with no single model showing clear dominance, though mistral and mixtral performed slightly below the others.

5 Related work

Varied methods have been adopted to generate OAS documents for Rest APIs. SpyREST (Sohan et al., 2015) employs an HTTP proxy server to intercept HTTP traffic to generate API documentation. Respector (Huang et al., 2024) employs static and symbolic program analysis to automatically generate OAS for REST APIs from their source code.

Similar to our approach, several studies have investigated converting parts of API documentation webpages into OAS. AutoREST (Cao et al., 2017) and captures part of the information presented in API documentation webpages and converts it into an OAS by a set of fixed rules. D2Spec (Yang et al., 2018) aims to extract base URLs, path templates, and HTTP method types, using rule-based web crawling techniques and classic machine learning to identify potential API call patterns in URLs. Bahrami and Chen (2020); Bahrami et al. (2020) combines rule-based and machine-learning algorithms to generate OAS from API documentation. They also develop a deep model to pinpoint fine-grained mapping of extracted API attributes to OAS objects. Most similar to our work, Andročec and Tomašić (2023) used GPT-3 to automatically generate OAS from a preprocessed HTML file describing an API documentation. OASBuilder distinguishes itself from their methodology by (1) dividing the generation task into multiple parts, and (2) extracting relevant information from webpages, thus accommodating long webpages that exceeds the context size of LLMs while breaking the task into more manageable subtasks for LLMs..

6 Conclusions

This paper presents OASBuilder, a novel multi-stage system designed to automatically generate and enhance OAS from online API documentation. By integrating rule-based algorithms with generative LLMs, OASBuilder addresses existing limitations in previous solutions. Our experiments demonstrate that OASBuilder is robust and capable of generalizing across hundreds of API documentation websites. Furthermore, a detailed evaluation reveals that the generated OAS captures most of the information from the documentation, significantly reducing the manual effort required of technical experts. The AI-based enhancement tools, combined with the UI platform, offer developers an end-to-end process yielding in a high-quality final result.

References

Darko Andročec and Matija Tomašić. 2023. Using gpt-3 to automatically create restful service descriptions. In *2023 4th International Conference on Communications, Information, Electronic and Energy Systems (CIEES)*, pages 1–4. IEEE.

Mehdi Bahrami, Mehdi Assefi, Ian Thomas, Wei-Peng Chen, Shridhar Choudhary, and Hamid R Arabnia. 2020. Deep sas: A deep signature-based api specification learning approach. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1994–2001. IEEE.

Mehdi Bahrami and Wei-Peng Chen. 2020. Automated web service specification generation through a transformation-based learning. In *Services Computing–SCC 2020: 17th International Conference, Held as Part of the Services Conference Federation, SCF 2020, Honolulu, HI, USA, September 18–20, 2020, Proceedings 17*, pages 103–119. Springer.

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*, pages 453–461. Springer.

Harrison Chase. 2022. Langchain.

Peter J. Danielsen and Alan Jeffrey. 2013. Validation and interactivity of web api documentation. In *2013 IEEE 20th International Conference on Web Services*, pages 523–530.

Paola Espinoza-Arias, Daniel Garijo, and Oscar Corcho. 2020. Mapping the web ontology language to the openapi specification. In *International Conference on Conceptual Modeling*, pages 117–127. Springer.

A Shaji George and AS Hovan George. 2023. A review of chatgpt ai's impact on several business sectors. *Partners universal international innovation journal*, 1(1):9–23.

Ruikai Huang, Manish Motwani, Idel Martinez, and Alessandro Orso. 2024. Generating rest api specifications through static analysis.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts. *Preprint*, arXiv:2401.04088.

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*, ISSTA 2022, page 289–301, New York, NY, USA. Association for Computing Machinery.

Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, Manish Sethi, Xuan-Hong Dang, Pengyuan Li, Kun-Lung Wu, Syed Zawad, Andrew Coleman, Matthew White, Mark Lewis, Raju Pavuluri, Yan Koyfman, Boris Lublinsky, Maximilien de Bayser, Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Yi Zhou, Chris Johnson, Aanchal Goyal, Hima Patel, Yousaf Shah, Petros Zerfos, Heiko Ludwig, Asim Munawar, Maxwell Crouse, Pavan Kapanipathi, Shweta Salaria, Bob Calio, Sophia Wen, Seetharami Seelam, Brian Belgodere, Carlos Fonseca, Amith Singhee, Nirmit Desai, David D. Cox, Ruchir Puri, and Rameswar Panda. 2024. Granite code models: A family of open foundation models for code intelligence. *Preprint*, arXiv:2405.04324.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik

Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Connor Shorten, Charles Pierse, Thomas Benjamin Smith, Erika Cardenas, Akanksha Sharma, John Trengrove, and Bob van Luijt. 2024. Structuredrag: Json response formatting with large language models. *Preprint*, arXiv:2408.11061.

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)*, pages 271–276. IEEE.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Mandana Vaziri, Louis Mandel, Avraham Shinnar, Jérôme Siméon, and Martin Hirzel. 2017. Generating chat bots from web api specifications. In *Proceedings of the 2017 ACM SIGPLAN international symposium on new ideas, new paradigms, and reflections on programming and software*, pages 44–57.

Jinqiu Yang, Erik Wittern, Annie T. T. 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*, MSR '18, page 454–464, New York, NY, USA. Association for Computing Machinery.

A Appendix

A.1 Example of an API documentation webpage

Figure 3 shows an example of a real-world API documentation webpage taken from Shopify API website.[6]

A.2 Descriptive Documentation Retrieval Algorithm

To find the descriptive documentation, we looked for an HTML scope in the webpage which encompasses this information, we call this scope "minimal ancestor". To that end, we employed two distinct approaches. In scenarios where a webpage incorporates multiple API calls, we defined the scope as the highest ancestor of the request example HTML element that does not encompass other requests [7]. In Figure 1, this scope should encompass both reference-based and example-style sections. Conversely, when dealing with a webpage containing a single API call, we conducted a search for *leaf elements.*[8] likely associated with parameters in the reference-based documentation based on their text, such as parameters from the request or response, and parameter header templates. These elements could be situated, for instance, in the "Parameters Description" section as illustrated in Figure 1. Subsequently, we iterated through the ancestors of each identified element, starting from the immediate parent and moving upwards, in search of the first ancestor containing a matching URL endpoint corresponding to the provided API URL. Since this is often found preceding the HTTP method (e.g. "GET /info/id"), we denote it as "HTTP Method" and "Endpoint/URL" in Figure 1.

After retrieving these minimal ancestors, we rank them according to two criteria: 1) the number of parameters from the request or response found as leaf elements in the ancestor, and 2) whether the HTTP method type of the URL was found as a leaf element. Following this ranking, we filter out HTML elements that are ancestors of other candidates. Lastly, if we still have multiple candidates sharing the same rank, we randomly sample one of them, although we did not encounter such cases in our experiments.

The minimal ancestor is then preprocessed to remove noise and tailor it to the constrained context size of the LLM. This involves filtering out its children that are less likely to contain relevant information for augmenting the base OAS. Specifically, we search for parameter names extracted from the API request/response example and syntactic hints such as the structure of an HTML parameters table. Additionally, we exclude the request and response examples at this stage, as they have already been utilized in generating the base OAS. Finally, all HTML attributes are removed, as they are deemed less likely to contain relevant information.

For a formalized presentation of the flow, see Algorithm 1

A.3 Prompt Generation Examples

In order to generate the OAS, we applied in-context learning where the inputs are preprocessed content found in the API documentation webpage, and the outputs are components from the OAS or partial OAS containing the input data. The in-context examples were chosen from real-world APIs (e.g., github API) while we tried to balance between the length and the diversity of the examples. In Figures 6, 4, 5 we provide examples of the prompts we used for this purpose. Due to space limitations, we have not included all the in-context examples, but we would be happy to share them upon request.

A.4 URLs for Base OAS Generation

- https://docs.sendgrid.com/api-reference/contacts/add-or-update-a-contact
- https://developer.servicenow.com/dev.do#!/reference/api/sandiego/rest/c_TableAPI
- https://dev.fitbit.com/build/reference/web-api/activity/get-activity-log-list/
- https://docs.adyen.com/api-explorer/Checkout/70/post/payments
- https://openweathermap.org/api/one-call-3
- https://developer.cisco.com/meraki/api-v1/get-device-camera-custom-analytics/

[6] https://shopify.dev/docs/api/admin-rest/2024-10/resources/inventorylevel

[7] If the minimal ancestor of the subsequent request is not consecutive, it is defined as a sequence of elements ending in the ancestor of the next request

[8] HTML elements lacking children

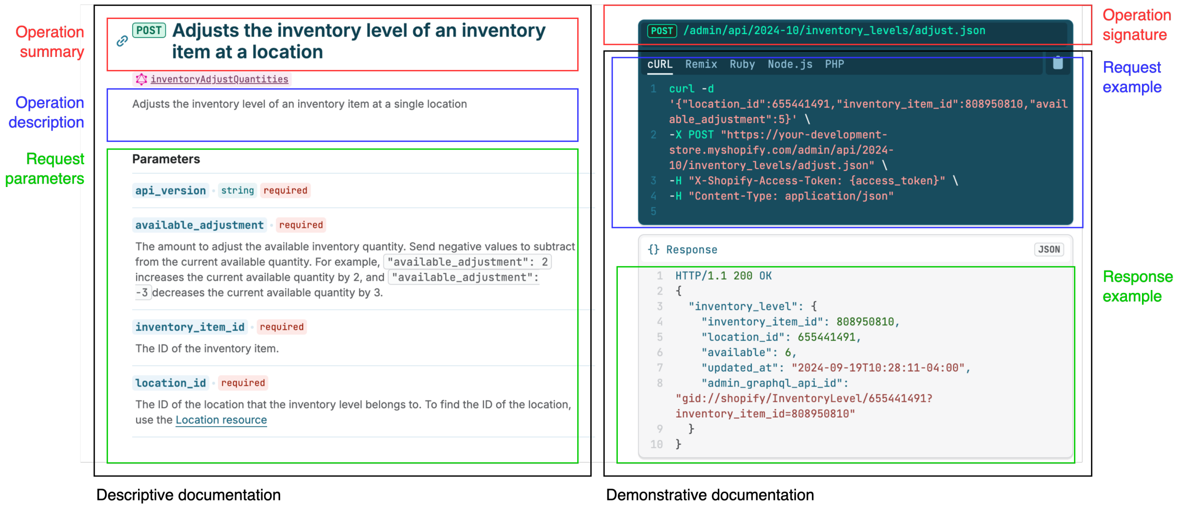

Figure 3: Example of an API documentation webpage taken from Shopify API in 25.11.2024 (taken from https://shopify.dev/docs/api/in-rest/2024-10/resources/inventorylevel.

Algorithm 1 Generate Descriptive Documentation

Input: Example of API Request, and API HTML Documentation
Output: HTML Element for Enrichment

function FINDMINIMALHTMLELEMENT(API_Spec, API_Doc, M)
  1. Extract parameter names from given API request.
  2. Find elements in documentation that their texts match a parameter name or a parameter header.
  3. For each candidate find first HTML elements which meets one of the following criteria:
    a. Contains an endpoint HTML element matching the API URL from the request.
    b. Contains multiple HTML elements of the same parameter name from the API specification.
    iv. Select the HTML element from the candidates by ranking according to the following criteria by the following order:
      1. Number of parameter names from the API specification found in its context by exact matching*.
      2. Whether an endpoint matching the URL was found.
      3. Whether the extracted HTTP method type was found by exact matching.
      4. Whether they contain a "table" HTML element.
      5. Minimality of scope (i.e. filtering out parents of candidates).
  Preprocessing the minimal HTML element:
    i. Iterate over the minimal HTML element children and filter according to the following criteria:
      1. Whether the child is a "table" HTML element.
      2. Whether the child is preceded by a parameter header HTML element.
      3. Whether the child contains any extract parameter name by exact matching.
      4. Whether the child contains the phrases "required" or "optional" by exact matching.
    ii. Remove all the attributes of the HTML elements.
  Generate Structured data from minimal HTML element:
    i. Apply M to generate a description of the API and a table where each row represents relevant metadata about a parameter found in the minimal HTML element's content. This can be achieved by techniques such as In-Context Learning, or by training a language model on a manually-labeled dataset.
  Integrate Generated Data into API Specification.
end function

Prompt instructions

```
Write an OpenAPI JSON specification for the following REST GET API. JSON object keys must be string literals.
```

In context example

Input:

```
curl -X GET \'https://api-m.sandbox.paypal.com/v1/payment-experience/web-profiles/XP-RFV4-PVD8-AGHJ-8E5J?
organization_id=10234695\' -H \'x-amz-data-partition: DirectoryArn\'
```

Output:


```
{ "openapi": "3.0.0", "info": { "title": "PayPal Web Profiles API", "version": "1.0.0", "description": "API for
managing web profiles in PayPal" }, "servers": [ { "url": "https://api-m.sandbox.paypal.com" } ], "paths": {
"/v1/payment-experience/web-profiles/{profileId}": { "get": { "summary": "Retrieve a web profile by ID",
"operationId": "get_web_profile_by_id", "tags": [ "Web Profiles" ], "parameters": [ { "name": "profileId",
"in": "path", "required": true, "schema": { "type": "string", "example": "XP-RFV4-PVD8-AGHJ-8E5J" },
"description": "The ID of the profile for which to show details." }, { "name": "organization_id", "in":
"query", "required": false, "schema": { "type": "integer", "example": "10234695" }, "description":
"Organization id" }, { "name": "x-amz-data-partition", "in": "header", "required": true, "schema": { "type":
"string", "example": "DirectoryArn" }, "description": "The Amazon Resource Name (ARN) that is associated with
the Directory where the object resides." } ], "responses": { "200":{ "description": "successful operation" } }
} } } }
```


Test example

Input:

```
curl -X GET \'https://your-development-store.myshopify.com/admin/api/2024-04/inventory_levels.json?
location_ids=655441491\' -H \'X-Shopify-Access-Token: {access_token}\'
```

Output (generated):


```
{ "openapi": "3.0.0", "info": { "title": "Shopify Inventory Levels API", "version": "2024-04" }, "servers": [ {
"url": "https://your-development-store.myshopify.com" } ], "paths": { "/admin/api/2024-
04/inventory_levels.json": { "get": { "summary": "Retrieve inventory levels for a specific location",
"operationId": "get_inventory_levels_by_location", "tags": [ "Inventory Levels" ], "parameters": [ { "name":
"location_ids", "in": "query", "required": true, "schema": { "type": "integer", "example": 655441491 },
"description": "IDs of the locations to retrieve inventory levels for" }, { "name": "X-Shopify-Access-Token",
"in": "header", "required": true, "schema": { "type": "string" }, "description": "X Shopify access token" } ],
"responses": { "200":{ "description": "successful operation" } } } } } }
```


Figure 4: Example prompt for generating an OAS from a cURL command. The prompt includes two in-context examples (only one is shown here for brevity). Information from the cURL command is extracted to create an OAS featuring a single operation, complete with a `title`, `version`, `servers`, `paths`, `operationId`, `tags`, and detailed `parameters`, including their `types`, `descriptions`, and `examples`.

Prompt instructions

```
Write a JSON schema for the following object. JSON object keys must be string literals. Make sure that all $ref
are defined under components/schemas. Use true, false as JSON booleans. Do not generate enums.
```

In context example

Input:


```
{ "firstName": "John", "lastName": "Doe", "age": 21, "tags": [], "assigned_to_user": null, "site_admin": false}
```


Output:


```
{ "type": "object", "properties": { "firstName": { "type": "string", "description": "The person\'s first
name.", "example": "John" }, "lastName": { "type": "string", "description": "The person\'s last name.",
"example": "Doe" }, "age": { "type": "integer", "description": "Age in years which must be equal to or greater
than zero.", "example": 21 }, "tags": { "type": "array", "description": "The tags associated with the person",
"items": { "type": "string" } }, "assigned_to_user": { "type": "object", "description": "User assigned to the
person" }, "site_admin": { "type": "boolean", "description": "Indicates whether this user is a site admin",
"example": false } }}
```


Test example

Input:


```
{ "inventory_levels": [ { "inventory_item_id": 49148385, "location_id": 655441491, "available": 2,
"updated_at": "2024-09-19T10:10:57-04:00", "admin_graphql_api_id": "gid://shopify/InventoryLevel/655441491?
inventory_item_id=49148385" } ]}
```


Output (generated):


```
{ "type": "object", "properties": { "inventory_levels": { "type": "array", "items": { "type": "object",
"properties": { "inventory_item_id": { "type": "integer", "example": 49148385, "description": "The inventory
item id" }, "location_id": { "type": "integer", "example": 655441491, "description": "The location id" },
"available": { "type": "integer", "example": 2, "description": "The available quantity" }, "updated_at": {
"type": "string", "example": "2024-09-19T10:10:57-04:00", "description": "The updated at date" },
"admin_graphql_api_id": { "type": "string", "example": "gid://shopify/InventoryLevel/655441491?
inventory_item_id=49148385", "description": "The admin graphql api id" } } } } }}
```


Figure 5: Prompt example to generate a JSON schema from a given JSON object or array. This prompt is used to generate both the requestBody and the responses which are later set in the corresponding OAS.

- https://developer.paypal.com/docs/api/payment-experience/v1/#web-profile_create
- https://stripe.com/docs/api
- https://developer.webex.com/docs/api/v1/meeting-transcripts/download-a-meeting-transcript
- https://developer.okta.com/docs/api/openapi/okta-management/management/tag/ApiServiceIntegrations/#tag/ApiServiceIntegrations/operation/activateApiServiceIntegrationInstanceSecret
- https://developer.okta.com/docs/api/openapi/okta-management/management/tag/ApplicationGroups/#tag/ApplicationGroups/operation/assignGroupToApplication
- https://learn.microsoft.com/en-us/linkedin/shared/integrations/communications/invitations?context=linkedin%2Fcompliance%2Fcontext&view=li-lms-unversioned&preserve-view=true
- https://www.aha.io/api/resources/ideas/create_an_idea
- https://www.reddit.com/dev/api
- https://cloud.ibm.com/apidocs/speech-to-text
- https://apidocs.orderdesk.com/?shell#create-an-order
- https://developer.atlassian.com/cloud/trello/rest/api-group-actions/#api-actions-idaction-reactions-post
- https://docs.github.com/en/rest/issues/comments?apiVersion=2022-11-28#create-an-issue-comment
- https://docs.github.com/en/rest/actions/workflow-runs?apiVersion=2022-11-28#re-run-a-job-from-a-workflow-run--code-samples
- https://developers.facebook.com/docs/whatsapp/business-management-api/guides/migrate-phone-to-different-waba

Prompt instructions

```
The following text contains HTML content that described a table of API parameters and request body properties. Each entry includes attributes such as name, type (string, int, etc.), required or optional status, and metadata (e.g., enum, default, max, format). Additionally, parameters have a location (path, header, query, cookie). Your task is to: 1. Parse the HTML to identify all request parameters and request body properties. 2. Extract all relevant information about each parameter and property. Extract metadata also from the descriptions. 3. Output an OAS document containing all the extracted information. Note: 1. Exclude parameters from the response. 2. Do not generate a components section in the OAS document. 3. Metadata can be found both inside and outside the descriptions. Here is an input-output example pair:
```

In context example

Input:

```
<div><h3><a>Parameters for "List issue comments for a repository"</a></h3><table><caption>Headers</caption><thead><tr><th>Name, Type, Description</th></tr></thead><tbody><tr><td><div><div><code>accept</code> <span>string</span> </div><div><div><p>Setting to <code>application/vnd.github+json</code> is recommended.</p></div><div></div></div></div></td></tr></tbody></table><table><caption>Path parameters</caption><thead><tr><th>Name, Type, Description</th></tr></thead><tbody><tr><td><div><div><code>owner</code> <span>string</span> <span>Required</span></div><div><div><p>The account owner of the repository. The name is not case sensitive.</p></div><div></div></div></div></td></tr>...</code></div></div></div>
```

Output:


```
{"openapi": "3.0.0", "info": {"title": "", "version": "1.0.0"}, "paths": {"/repos/{owner}/issues/comments": {"post": {"responses": {"200": {"description": "Success"}}, "parameters": [{"name": "owner", "description": "The account owner of the repository. The name is not case sensitive.", "in": "path", "required": true, "schema": {"type": "string"}}, ...]}}}
```


Test example

Input:

```
<div><div><div>
<div><div><h2><div><div><span>get</span></div><div><span>Retrieves a list of inventory levels</span></div>
</div></h2></div></div>
<div><div><a><span></span><span>inventoryItem</span></a></div></div>
<div><div><p>Retrieves a list of inventory levels.</p>
<p>You must include <code>inventory_item_ids</code>, <code>location_ids</code>, or both as filter parameters.
</p><p><strong>Note:</strong> This endpoint implements pagination by using links that are provided in the response header. To learn more, refer to <a>Make paginated requests to the REST Admin API</a>.</p></div></div>
<div><div><h3><span>Parameters</span></h3></div></div>
<div><div><div><div>api_version</div><span></span><div><span>string</span></div><div><span>required</span>
</div></div></div><hr/></div>
<div><div><div><div>inventory_item_ids</div><span></span><div><span>≤ 50</span></div></div><div><div>A comma-separated list of inventory item IDs. To find the ID of an inventory item, use the <a>Inventory Item resource</a></div></div></div><hr/></div>
<div><div><div><div>limit</div><span></span><div><span>≤ 250</span></div><div><span>default 50</span></div>
</div><div><div>The maximum number of results to show.</div></div></div><hr/></div>
<div><div><div><div>location_ids</div><span></span><div><span>≤ 50</span></div></div><div><div>A comma-separated list of location IDs. To find the ID of a location, use the <a>Location resource</a></div></div>
</div><hr/></div>
<div><div><div><div>updated_at_min</div></div><div><div>Show inventory levels updated at or after date (format: 2019-03-19T01:21:44-04:00).</div></div></div><hr/></div>
...
</div></div></div>
```

Output (generated):


```
{ "openapi": "3.0.0", "info": { "title": "", "version": "1.0.0" }, "paths": { "/admin/api/2024-04/inventory_levels.json": { "get": { "responses": { "200": { "description": "Success" } }, "parameters": [ { "name": "api_version", "in": "query", "required": true, "schema": { "type": "string" } }, { "name": "inventory_item_ids", "description": "A comma-separated list of inventory item IDs. To find the ID of an inventory item, use the Inventory Item resource", "in": "query", "required": false, "schema": { "type": "string", "maxLength": 50 } }, { "name": "limit", "description": "The maximum number of results to show.", "in": "query", "required": false, "schema": { "default": 50, "maximum": 250, "type": "integer" } }, { "name": "location_ids", "description": "A comma-separated list of location IDs. To find the ID of a location, use the Location resource", "in": "query", "required": false, "schema": { "type": "string", "maxLength": 50 } }, { "name": "updated_at_min", "description": "Show inventory levels updated at or after date (format: 2019-03-19T01:21:44-04:00).", "in": "query", "required": false, "schema": { "type": "string", "format": "date-time" } } ] } } }}
```


Figure 6: An example of a prompt used for generating an OAS based on the descriptive documentation found in the API documentation webpage. The model extract the relevant information from the HTML elements, and sets the fields' `description`, `type`, `required`, `enum`, and `format` metadata properties.

- https://docs.github.com/en/rest/issues/issues?apiVersion=2022-11-28
- https://community.workday.com/sites/default/files/file-hosting/restapi/index.html
- https://community.workday.com/sites/default/files/file-hosting/restapi/index.html
- https://community.workday.com/sites/default/files/file-hosting/restapi/index.html#budgets/v1/post-/runBudgetCheck
- https://docs.sendgrid.com/api-reference/contacts/delete-contacts
- https://docs.sendgrid.com/api-reference/custom-fields/update-custom-field-definition
- https://docs.sendgrid.com/api-reference/custom-fields/create-custom-field-definition
- https://shopify.dev/docs/api/admin-rest/2023-04/resources/asset#put-themes-theme-id-assets
- https://shopify.dev/docs/api/admin-rest/2023-04/resources/product#put-products-product-id
- https://docs.mapbox.com/api/search/geocoding/
- https://developers.facebook.com/docs/whatsapp/business-management-api/message-templates
- https://wit.ai/docs/http/20230215/#post__utterances_link
- https://www.twilio.com/docs/sms/api/deactivations-resource
- https://www.twilio.com/docs/sms/api/media-resource
- https://www.twilio.com/docs/sms/api/message-resource#read-multiple-message-resources
- https://airtable.com/developers/web/api/delete-multiple-records
- https://airtable.com/developers/web/api/update-record
- https://airtable.com/developers/web/api/refresh-a-webhook
- https://developer.cisco.com/meraki/api-v1/blink-device-leds/
- https://developer.cisco.com/meraki/api-v1/get-network-events/
- https://developer.cisco.com/meraki/api-v1/get-organization-summary-top-appliances-by-utiliz
- https://docs.github.com/en/free-pro-team@latest/rest/billing/billing?apiVersion=2022-11-28#get-github-actions-billing-for-an-organization
- https://docs.github.com/en/rest/issues/comments?apiVersion=2022-11-28
- https://docs.github.com/en/rest/interactions/user?apiVersion=2022-11-28
- https://docs.github.com/en/rest/search/search?apiVersion=2022-11-28
- https://learn.microsoft.com/en-us/linkedin/shared/api-guide/concepts/pagination?context=linkedin%2Fconsumer%2Fcontext
- https://dev.fitbit.com/build/reference/web-api/sleep/delete-sleep-log/
- https://dev.fitbit.com/build/reference/web-api/body/create-bodyfat-log/
- https://dev.fitbit.com/build/reference/web-api/friends/get-friends-leaderboard/

A.5 Examples of Generated Descriptions and Examples

Figure 7 and Figure 8 are respectively examples of generated descriptions and examples from the enhancements described in Section 3.4.

```json
{
    "200": {
        "content": {
            "application/json": {
                "schema": {
                    "properties": {
                        "dateLastActivity": {
                            "type": "string",
                            "description": "The date the activity was last updated."
                        },
                        "dateLastView": {
                            "type": "string",
                            "description": "The last time the user viewed the board."
                        },
                        "idTags": {
                            "type": "string",
                            "description": "A comma-separated list of tag IDs. Only actions within
                            ↪  these tags will be returned."
                        }
                    }
                }
            }
        }
    }
}
```

Figure 7: Example of enhancement for generating descriptions. Added lines are highlighted in green. Original documentation page for OAS is https://developer.atlassian.com/cloud/trello/rest/api-group-actions/#api-actions-idaction-reactions-post.

A.6 Most Popular URLs by Postman

To further establish the claim that most real-world APIs do not publish API specification. We manually checked whether the most popular APIs according to Postman[9] published an API specification in their API documentation webpages. We found that only five out of the fourteen contained OAS. The full findings are detailed in Table 3

[9]https://www.postman.com

```
{
    "parameters": [
        {
            "name": "owner",
            "in": "path",
            "required": true,
            "schema": {
                "type": "string",
                "example": "octocat",
                "x-ibm-examples": [
                    "hubot",
                    "other_user"
                ]
            },
           "description": "The account owner of the repository. The name is not case sensitive.",
            "x-ibm-grounded-description": true
        },
        {
            "name": "repo",
            "in": "path",
            "required": true,
            "schema": {
                "type": "string",
                "example": "octocat/Hello-World",
                "x-ibm-examples": [
                    "octocat/Spoon-Knife",
                    "octocat/hello-world"
                ]
            },
           "description": "The name of the repository without the \".git\" extension. The name is
            ↪ not case sensitive.",
            "x-ibm-grounded-description": true
        }
    ]
}
```

Figure 8: Example of enhancement for generating examples. Added lines are highlighted in green. Original documentation page for OAS is https://docs.github.com/en/rest/issues/comments?apiVersion=2022-11-28#create-an-issue-comment.

Site	API Documentation URL	Contains OAS
Salesforce	https://developer.salesforce.com/docs/apis#browse	No
Microsoft Graph	https://learn.microsoft.com/en-us/graph/overview	No
Slack	https://api.slack.com/docs/apps	No
PayPal	https://developer.paypal.com/api/rest/	No
Zoho CRM	https://www.zoho.com/crm/developer/docs/api/v7/modules-api.html	No
Cisco Meraki	https://developer.cisco.com/meraki/	Yes
Pipedrive API	https://developers.pipedrive.com/docs/api/v1	Yes
Amplitude	https://amplitude.com/docs/apis/analytics	No
BookingAPI	https://developers.booking.com/demand/docs	Yes
Amadeus	https://developers.amadeus.com/self-service	Yes
Symbl	https://docs.symbl.ai/reference	No
Hyperledger Besu	https://besu.hyperledger.org/stable/public-networks/reference/api	No
PingOne	https://apidocs.pingidentity.com/pingone/platform/v1/api/	No
Lob	https://docs.lob.com/	Yes

Table 3: Comparison of the most popular APIs on Postman for 2023, indicating whether they publicly publish their OAS (based on https://www.postman.com/explore/most-popular-apis-this-year).