APIstic: A Large Collection of OpenAPI Metrics

Souhaila Serbout
Software Institute, USI
Lugano, Switzerland
souhaila.serbout@usi.ch

Cesare Pautasso
Software Institute, USI
Lugano, Switzerland
c.pautasso@ieee.org

ABSTRACT

In the rapidly evolving landscape of web services, the significance of efficiently designed and well-documented APIs is paramount. In this paper, we present APIstic an API analytics dataset and exploration tool to navigate and segment APIs based on an extensive set of pre-computed metrics extracted from OpenAPI specifications, sourced from GitHub, SwaggerHub, BigQuery and APIs.guru. These pre-computed metrics are categorized into structure, data model, natural language description, and security metrics. The extensive dataset of varied API metrics provides crucial insights into API design and documentation for both researchers and practitioners. Researchers can use APIstic as an empirical resource to extract refined samples, analyze API design trends, best practices, smells, and patterns. For API designers, it serves as a benchmarking tool to assess, compare, and improve API structures, data models, and documentation using metrics to select points of references among 1,275,568 valid OpenAPI specifications. The paper discusses potential use cases of the collected data and presents a descriptive analysis of selected API analytics metrics.

The dataset available at: http://openapi.inf.usi.ch/

ACM Reference Format:
Souhaila Serbout and Cesare Pautasso. 2018. APIstic: A Large Collection of OpenAPI Metrics. In *Proceedings of 21st International Conference on Mining Software Repositories (MSR 2024).* ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

API analytics aims to obtain actionable insights from API-related artifacts (e.g., developer documentation [65], usage logs [17, 44], or – as considered in this paper – machine-readable descriptions [58]). API providers analyze their API landscape to improve the quality [57, 64] and value proposition of their APIs [56]. API consumers need support when selecting suitable APIs [27] to estimate whether the risk of introducing an extra dependency is worth the benefit of integrating their systems with some external API [37, 53].

In this paper, we present a dataset of API analytics metrics belonging to 1,275,568 valid API Specifications obtained by mining a large collection of real-world Web APIs [66] gathered from different sources. We include metrics covering different facets of Web APIs [37, 46] described according to the widely adopted standard specification: OpenAPI [8, 27]. API Structure Metrics, e.g., the number of paths, operations, HTTP methods, and distinct parameters, offer insights into the size and type of the provided API operations. The API Data Model Metrics, counting schemas, properties, and their usage, delve into the intricacies of data representation in APIs. The API Description Metrics, such as endpoints description coverage, the Coleman–Liau index (CLI) and Automated Readability Index (ARI), provide an evaluation of API natural language documentation quality. API Security metrics assess whether, how and to which extent an API supports client authentication and authorization [24].

The dataset provides a benchmark for API designers to assess quantitative aspects of their API in relationship to a large collection of API descriptions mined from open source repositories (e.g., as shown in this threshold derivation method [20]). API analytics researchers can also use the dataset to quantitatively observe the state of the practice in API design, study to which extent some known API design patterns (or smells) are adopted in practice [74], select and extract smaller samples based on quantitative and domain-specific criteria for further study, identify new indicators based on the given raw metrics that can help detect outliers or cluster similar API designs, making it possible to assess and manage the quality of entire API landscapes [51, 70]. By including artifacts obtained from different sources, the dataset includes APIs at different stages of their development lifecycle: from early API sketches found in GitHub repositories to mature APIs deployed in production by major service providers. Thanks to the corresponding interactive exploration tool, researchers to search and filter our comprehensive API analytics dataset for empirical studies and pattern mining [40]. We aim to regularly update this public dataset with new features.

To demonstrate the dataset's utility, this paper includes a descriptive analysis of key API metrics, comparing artifacts from various sources. We address questions about the variability of the metrics over time and across datasets. For example, we observe the usage of specific HTTP methods and API security features as well as measure the readability of natural language documentation.

The rest of this paper is structured as follows. In Section 2 we describe how the dataset has been gathered, the provenance metadata associated with the artifacts and give an informal definition of the metrics. In Section 3, we present the results of the analysis over the dataset, with the goal of comparing the artifacts obtained from different sources as seen through a multifaceted lens of the metrics. We discuss potential use cases for the dataset in Section 4. Then we introduce some related work in Section 5 before we draw our conclusions in Section 6 where we also outline the roadmap for the APIstic dataset.

MSR 2024, April 2024, Lisbon, Portugal
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06...$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 DATASET DESCRIPTION

2.1 Data Collection Methodology

The retrieval of API specifications varies by source. APIs.guru and SwaggerHub, specializing in OpenAPI specifications, have a lower risk of false positives. However, extracting from SwaggerHub is challenging due to API limitations [61]. For GitHub and BigQuery, we used a content-based discovery approach, targeting key elements of OpenAPI/Swagger specifications to reduce false positives. Discovered specifications are regularly processed for inclusion in future dataset updates.

2.1.1 SwaggerHub API. SwaggerHub [11], a platform for teams to design, document, and test HTTP-based web services. It feature also the SwaggerHub Registry API [12]. This API facilitates integration of API documentation management into projects, bypassing the web interface for easier automation of API management. The API offers also endpoints to navigate not only the owned APIs but also specifications that are publicly hosted on the SwaggerHub repository.

SwaggerHub, while lacking explicit documentation on its rate limiting strategy, seems to block clients exceeding 10 requests per second within a 3-minute window. Based on our observations, to reset this limit, clients must remain inactive for at least an hour. Consequently, collecting artifacts from the SwaggerHub dataset spanned several weeks. Additionally, due to frequent breaking changes in the SwaggerHub Registry API, our crawler necessitated continuous monitoring and maintenance.

2.1.2 GitHub API. GitHub is a popular source to collect artifacts found in open-source projects [28], including OpenAPI specifications. There is no established approach to fetch the OpenAPI artifacts using the GitHub APIs since they are stored as .yaml or .json files. However, since the GitHub API supports searching for files by content, we built two queries that look for any .json or .yaml file that contains the mandatory fields in a valid OpenAPI/Swagger specification:

q_1 = `language:yaml language:json "openapi" "paths" "info" "title"`

q_2 = `language:yaml language:json "swagger" "paths" "info" "title"`

GitHub's REST and GraphQL APIs have rate limits – 5,000 requests per hour for REST and 2,000 points per minute for GraphQL – slowing down data collection. However, these APIs are free, with no fees for requests or queries, unlike data stores like BigQuery.

Listing 1: Query to retrieve OpenAPI specifications from the BigQuery GitHub Archive

```
SELECT f.id, f.repo_name, f.path, f.ref, c.content, c.size
FROM `bigquery-public-data.github_repos.files` AS f
JOIN `bigquery-public-data.github_repos.contents` AS c
ON f.id = c.id
WHERE ((f.path LIKE '%.json' OR f.path LIKE '%.yml'
    OR f.path LIKE '%.yaml')
AND (c.content LIKE '%swagger%' OR c.content LIKE '%openapi%')
AND (c.content LIKE '%paths%'))
AND c.content IS NOT NULL
AND c.content != ''
LIMIT 1000 OFFSET ${PAGE};
```

2.1.3 BigQuery API. BigQuery is a cloud-based data warehouse provided by Google that allows users to store and analyze large datasets using SQL-like queries [32, 45]. The GitHub Archive [7] on BigQuery, recording all public GitHub commits since 2011, allows for retrieving snapshots of GitHub data using an SQL-like syntax. We fetched the open API specification using the query in Listing 1.

One disadvantage of using BigQuery to retrieve GitHub data is that it requires a Google Cloud Platform account, which may involve additional costs depending on the size of the dataset and the complexity of the queries. In the case of GitHub Archive, the first terabyte of data processed per month is free, but additional data processing incurs a fee of $5 per terabyte.

2.1.4 APIs.guru. APIs.guru [3] is a popular open-source project that provides a curated collection of OpenAPI specifications. It is a community-driven project that aims to provide a centralized repository that can be used by developers, service providers, researchers, and other stakeholders. Because of its relatively small size compared to the amount of API descriptions that can be found across other public data sources, it cannot be used standalone for large-scale studies. However, because of its curated content it contains high-quality, valid and distinct specifications as "non-reliable" APIs are filtered out.

The APIs.guru website includes a simple API which makes it easy to retrieve the whole dataset after enumerating its content, which greatly simplifies the discovery and retrieval of the specifications.

Table 1: Datasets Overview (41.03 GB Total Size)

	Number of API Specifications		
Source	Valid	Distinct	avg. size (kB)
GitHub [6]	887,775	505,199	203.97
SwaggerHub [12]	378,275	345,731	27.08
BigQuery [5]	6,376	6,198	443.38
APIs.Guru [3]	3,142	3,142	241.37
Combined	1,275,568	856,259	60.31

2.2 Dataset Overview

In Table 1, we provide an extensive analysis of the origins of the valid OpenAPI specifications in our dataset, with a significant portion originating from GitHub.

During the data collection process, we identified instances of identical API specifications sourced from different URLs, marked as duplicates in our collection. Despite this, we opted to keep these duplicates to facilitate code cloning research and track potential divergences in identical specifications over time.

Figure 1 offers a Venn diagram that quantifies the distinct and overlapping specifications among the GitHub, SwaggerHub, APIs.guru, and BigQuery datasets. This diagram underscores the individual and collective contributions of each source where 4011 identical API specifications were extracted from more than one source.

2.2.1 SwaggerHub Dataset. Until January 15th, 2024, our SwaggerHub crawler discovered URLs of 501,064 OpenAPI/Swagger specifications attributed to 275,929 unique owners (SwaggerHub user accounts). We downloaded and processed till today 432,265 specifications, out of which 378,275 were found to be valid and written in 12 different version of the specification language (2.0 $\longrightarrow$ 3.1.5). These specifications were developed on the platform over a span of nine years (Figure 2).

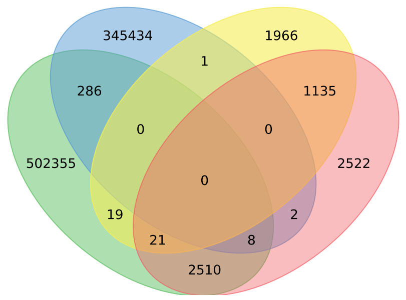


Figure 1: Number of overlapping API specifications
🟩 GitHub | 🟦 SwaggerHub | 🟨 APIs.Guru | 🟥 BigQuery

2.2.2 *GitHub Dataset.* The GitHub crawler discovered and collected 1,221,224 valid API descriptions as commits belonging to 469,029 APIs Until now we parsed, and bundled 887,775 artifacts with computed metrics present in the GitHub Dataset. These artifacts represent commits belonging to the history of 269,082 APIs committed by 92,250 different owners. 182,902 of the APIs have only one commit. One API has a long history with 594 commits.

2.2.3 *BigQuery Dataset.* The BigQuery crawler sent 430 paginated queries, limiting the size of response to 1000 entries per page. The average number of valid specifications found in each query's response was 14, overall producing 6,376 valid OpenAPI descriptions retrieved from 4,696 distinct repositories.

2.2.4 *APIs.guru Dataset.* We retrieved 3142 valid and distinct API specifications from the APIs.guru curated website on January 15th, 2024. While most overlap with the ones discovered through the other sources, we kept this dataset as it represents one of the largest manually curated API directories where high quality OpenAPI specifications can be found. For this reason it is a popular source for recent API analytics studies [15, 25, 42, 49].

2.3 Data Validation Methodology

All the API specifications included in our dataset are valid ones, regardless of their source or retrieval method. As they are fetched from their URL source, API specifications are checked using Swagger Validator [13], a popular Node.js library, which parses and validates their JSON or YAML content against versions 2 or 3 of the OpenAPI Standard Specification [8]. Only those the syntactically and semantically conform to the standard are retained in the dataset. We found that only the specifications from APIs.guru are all valid (Table 1).

After the initial validation, the API specifications are bundled into a single document following and traversing references to specification fragments found across different files. Specifications which cannot be successfully bundled due to broken links are also discarded. The files used to produce the APIstic metrics dataset are produced as a result of these validation and bundling processes.

As shown in Table 1 and Figure 1, we preserve duplicates to support code cloning studies. Likewise, we have found that some API specifications that appear to be empty (with no paths) have been used to model JSON schemas [21] (without any operations) or event-driven APIs which make use of webhooks to deliver notifications by calling back subscribers [18, 22].

2.4 Artifacts Provenance Metadata

Artifacts in our dataset are uniquely identified and carry explicit provenance metadata, so that a **URL** of their source is recorded. We also include timestamps obtained from their source as well as the **Fetching Date** timestamp tracking when the artifact was fetched by our crawler to be analyzed. The **Deleted** boolean flag indicates whether the description was still present (or not) at its URL when it was last fetched. Given the differences in the mechanism used to access each source, we also store the following source-specific metadata.

2.4.1 *GitHub.* For Each artifact we retrieve the following metadata:
• **File Name and Path**: The name and path to locate the API file within the repository.
• **Owner and Repository Name**: key identifiers in the vast data lake of GitHub repositories.
• **Commit Date**: Indicates when was the API description committed to the repository.
• **Commit SHA**: The hash value used to uniquely identify specific commits within a repository.

2.4.2 *BigQuery.* While BigQuery extracts API descriptions from the GitHub Archive, it unfortunately does not provide access to the same metadata. In particular, while the **File Name and Path** and **Owner and Repository Name** are present, there is no **Commit Date** or **Commit Hash**. Additionally, while descriptions sourced from the BigQuery API can belong to arbitrary branches – so we store their **Branch Name** – the ones fetched from GitHub are currently only fetched from the default branch.

2.4.3 *SwaggerHub.* From SwaggerHub, we obtain the:
• **Owner**: Identifies the creator or the original author of the API.
• **Creation Date**: Indicates the creation time of the API in SwaggerHub.
• **Modification Date**: Denotes the last time the API was altered and saved in SwaggerHub, useful for tracking updates and detecting changes.

2.4.4 *APIs.guru.* The API descriptions sourced from the APIs.guru directory have been curated by the APIs.guru maintainers with additional metadata including the API provider contact information, a category, and a provenance link to the original OpenAPI specification made available by the provider. This information is embedded in the API description itself. While APIs are manually classified within a set of 35 categories, the specifications lack explicit creation or modification timestamps providing a historical perspective over the content of the collection.

2.5 Versioning Metadata

OpenAPI specifications must include a **Version Identifier**, which we include in the metadata together with its classification according to two different criteria:
• **Version Identifier Format**: Categorizes the API version information provided in the info section, or API endpoints. It distinguishes whether the API developers employ semantic versioning [10] (with two or three counters), time-based versioning, numeric versioning, or other versioning schemes [46, 59].

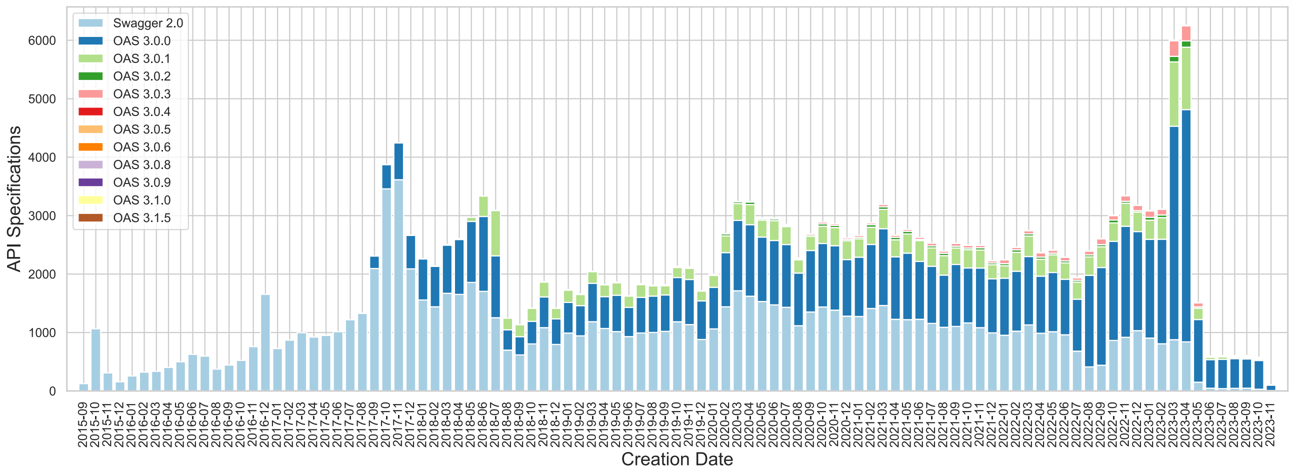


Figure 2: Monthly Distribution by Creation Date of Valid OpenAPI/Swagger Specifications in SwaggerHub Dataset

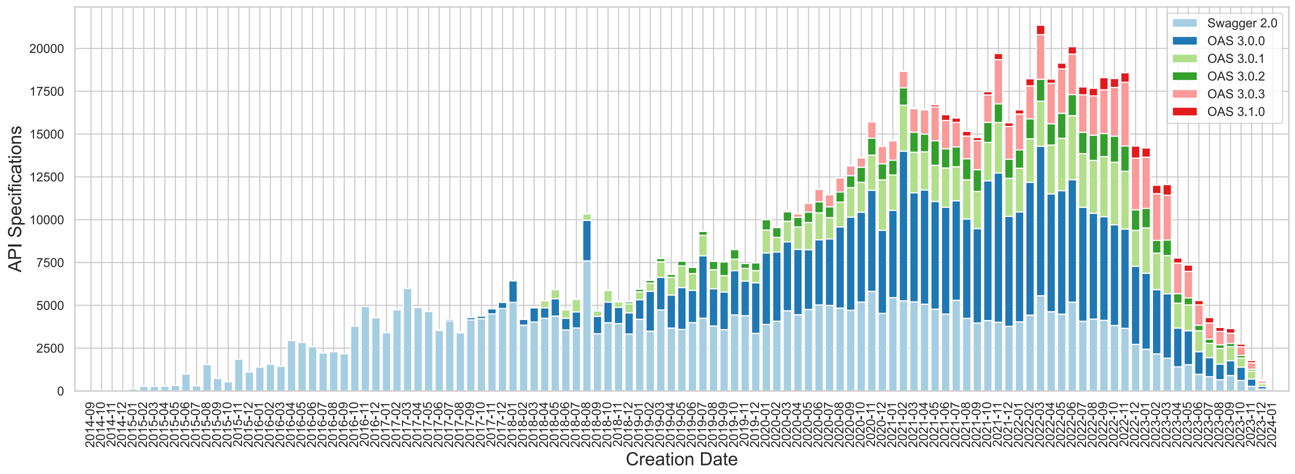


Figure 3: Monthly Distribution of Commits of Valid OpenAPI/Swagger Specifications in GitHub Dataset

• **Release type**: Categorizes the type of release: whether the API artifact describes a stable release or different kinds of preview releases. This can be useful to refine the API specifications based on their maturity level [48].

2.6 Metrics

Building upon the static analysis metrics proposed by Bogner et al., Haupt et al., Serbout et al. in [19, 36, 60] to study the size and complexity of API structures, the metrics in our dataset share the goal of providing a quantitative assessment of the size of API structures and data models, but also of selected quality attributes of API specifications, including their complexity, readability, versioning and security [24].

Metrics are computed starting from bundled OpenAPI specifications by running custom analytics code. Some metrics can be computed directly by running database queries, while others require parsing and processing the OpenAPI specifications with custom analytics scripts. The scripts we built to compute the metric and classifications are all available in: https://anonymous.4open.science/r/APIstics_metrics-10C6/README.md.

2.6.1 *API Structure Metrics.*

These metrics evaluate the size, complexity of the operational features of the API, providing insights into its functional scope and diversity.

• **Paths**: The number of paths in the API. This metric indicates the breadth of the API's functionality, with each path representing the address of a different communication endpoint, resource or service provided by the API.
• **Operations**: The total count of operations available in the API. This reflects the API's operational capabilities, encompassing all possible actions that can be performed through it.
• **Used Methods**: The number of distinct HTTP methods (GET, POST, PUT, DELETE, etc) used across the API operations. It signifies the diversity in the API's interaction modes.
• **Parametric Operations**: The number of operations that use path or query parameters. This metric helps in understanding the complexity and customization potential of the API operations.
• **Distinct Parameters**: The count of unique parameter names used across the API. It representing the variety of parameters that the API can accept, reflecting its versatility.

• **Used Parameters**: The total number of times parameters are used in the API. This indicates how frequently the API relies on parameterization for its operations.

2.6.2 API Data Model Metrics. This set of metrics delves into the structure and usage of data models within the API, highlighting the size and complexity of its data representation.

• **Defined Schemas**: To gauge the size of the API data model we count the number of schemas defined in the API description.

• **Distinct Used Schemas**: The total number of distinct schemas that are actually mentioned in API request or response messages. This reflects the overlap between the theoretically provided API data model and the API data model clients can use in practice.

• **Properties**: The total count of properties within those schemas represents the granularity and detail of the data models used.

• **Used Properties**: The number of properties that are explicitly used as part of API request or response messages. It indicates to which extent the data model of the API is usable by API clients.

• **Distinct Used Properties**: The unique property names count indicates the diversity of data attributes the API handles.

2.6.3 API Natural Language Descriptions Metrics. These metrics focus on the quality and thoroughness of the API's natural language documentation augmenting its machine-readable, structured description [29]. APIs with extensive natural language descriptions can be considered as high priority candidates to use as inputs for machine learning models for clustering or classification tasks.

The OpenAPI specifications, which are machine-readable, can contain natural language descriptions in multiple languages. Out of many possible metrics to assess the readability of natural language [14, 26, 62], to quantitatively compare the expressiveness of these descriptions across various languages, we employed two widely recognized, language-independent indices. The Automated Readability Index (ARI) was chosen for its simplicity, easy computation, suitability for a range of texts, and language-agnostic nature. In contrast, the Coleman–Liau Index (CLI) is preferred for technical texts due to its emphasis on letter count, and it is notable for its simplicity, making it easily understandable.

• **Coleman-Liau Index**: A readability index computed based on readability formula designed to gauge the understandability of a text based on its characters per word and sentences per 100 words [26].

CLI is particularly suited for technical documents like API documentation, as it focuses on characters and sentences rather than syllables, which are more challenging to accurately assess in technical language [72].

For each API, we compute the following two metrics:

- **mccphw**: Mean Character Count per Hundred Words.
- **mscphw**: Mean Sentence Count per Hundred Words.

The average mean sentence count per hundred words (*Average mscphw*) is calculated as the total mean sentence count per hundred words for each path divided by the total number of paths. Mathematically, it can be represented as:

$$avg\_mscphw = \frac{\sum(\text{mscphw for each path})}{\text{Total number of paths}} \tag{1}$$

Similarly, the average mean character count per hundred words (*Average mccphw*) is computed as the mean character count per hundred words for each path divided by the total number of paths:

$$avg\_mccphw = \frac{\sum(\text{mccphw for each path})}{\text{Total number of paths}} \tag{2}$$

The Coleman–Liau index is then calculated using the formula:

$$\text{Index} = (0.0588 \times avg\_mccphw) - (0.296 \times avg\_mscphw) - 15.8 \tag{3}$$

• **Automated Readability Index**: A readability index that estimates the understandability of a text based on its character, word, and sentence counts [62]. This index provides an estimate of the US grade level needed to comprehend the text. It is formulated as follows:

$$\text{ARI} = 4.71 \times \frac{\text{characters}}{\text{words}} + 0.5 \times \frac{\text{words}}{\text{sentences}} - 21.43 \tag{4}$$

Where the characters, words, and sentences are the average counts per API endpoint.

• **Endpoints Description Coverage**: This is a percentage value indicating the proportion of API endpoints that include a non-empty description. It measures whether and to which extent all API endpoints have been completely documented.

2.6.4 API Security Metrics. This section assesses the security protocols and strategies implemented in the API, reflecting its overall security posture. OpenAPI provides explicit support for client authentication and authorization schemes such as API keys, the OAuth2 protocol, or HTTP basic authentication. Developers can customize how such schemes are mapped to the HTTP protocol request and response payloads and indicate in which endpoint they are employed.

• **Security Schemes**: The number of security schemes of each type listed in the API security component.

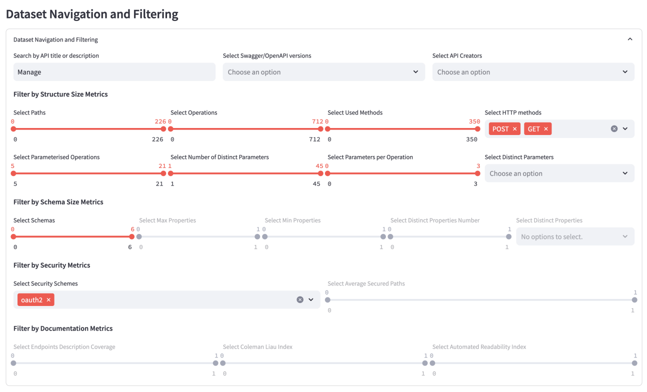
• **Secured Endpoints**: The number of API endpoints which explicitly employ specific security schemes.

2.7 APIstic Web-based Dataset Exploration Tool

APIstic capitalizes on the advanced capabilities of MongoDB for storing and managing the diverse range of API specifications. The choice of MongoDB is motivated by its exceptional handling of large datasets and its flexibility in accommodating the tree structures inherent in API specifications. As they are fetched from their sources, the stored API specifications are then retrieved and systematically analyzed by scripts that we meticulously designed to compute the previously defined metrics and classifications for each API artifact. Following the computation, the analytical outcomes for each API specification are persisted in MongoDB and made accessible by APIstic's Web Application, which provides a convenient UI for data filtering and navigatio (Figure 4).

3 API DATASET ANALYSIS

Our analysis encompasses various facets as informed by the computed metrics, including the growth and evolution of API structures and data models, and the temporal stability of these metrics. We delve into the utilization of HTTP methods within APIs, tracking their evolution. Furthermore, we investigate the security schemes in use and the degree of security applied to endpoints. Additionally, we evaluate API documentation readability by analyzing natural



Dataset Navigation and Filtering

Dataset Navigation and Filtering

Search by API title or description: Manage

Select Swagger/OpenAPI versions: Choose an option

Select API Creators: Choose an option

Filter by Structure Size Metrics

Select Paths: 0 – 226

Select Operations: 0 – 712

Select Used Methods: 0 – 350

Select HTTP methods: POST, GET

Select Parameterised Operations: 5 – 21

Select Number of Distinct Parameters: 1 – 45

Select Parameters per Operation: 0 – 3

Select Distinct Parameters: Choose an option

Filter by Schema Size Metrics

Select Schemas: 0 – 6

Select Max Properties: 0 – 1

Select Min Properties: 0 – 1

Select Distinct Properties Number: 0 – 1

Select Distinct Properties: No options to select.

Filter by Security Metrics

Select Security Schemes: oauth2

Select Average Secured Paths: 0 – 1

Filter by Documentation Metrics

Select Endpoints Description Coverage: 0 – 1

Select Coleman Liau Index: 0 – 1

Select Automated Readability Index: 0 – 1

Figure 4: API Metrics Filters in the APIstic Web UI

Table 2: Version formats in API specifications

Format	SwaggerHub	BigQuery	GitHub	APIs.guru
Semantic Versioning	346,012	3,325	546,581	1,161
Major version number	25,139	1,897	248,747	512
Date	1,059	223	18,423	780
Develop	160	68	1,513	2
Snapshot	977	67	6,310	1
Preview	179	48	1,089	489
Alpha	412	45	3,117	45
Beta	456	272	4,448	132
Release Candidate	370	12	1,211	0
Other	2,965	333	21,481	19
Tag	266	58	509	1
Not versioned	280	28	34,346	0

language descriptions with readability indices. The objective is to demonstrate the extensive scope and detail of our dataset, highlighting its value to API development researchers. The dataset's broad range of metrics acts as customizable filters, enabling the isolation of particular data segments for varied research needs.

In the subsequent distribution plots (box plots and violin plots), we have excluded outlier instances. We categorize a metric value as an outlier if it appears fewer than 10 times within a dataset.

3.1 Dataset Growth Over Time

We present in Figures 2 and 3, the monthly distribution of OpenAPI Specifications created on both SwaggerHub and GitHub platforms. The datasets have shown consistent monthly growth. As the OpenAPI 3.0 standard was introduced, specifications adhering to it began to surface, allowing the monitoring of its adoption. Despite this, both Swagger and OpenAPI remain in use, with over three years passing before the newer standard overshadowed the older. The decline in specifications gathered in 2023 is attributed to the ongoing data mining process at the time of this report.

3.2 Web API versioning

What are the commonly adopted version formats in Web APIs? In Table 2 we present an overview of the distribution of version identifier formats classified across the four data sources. The results show a predominant adoption of Semantic Versioning across all the data sources.

Dataset	Max	Min	Average	Median	StDev
GitHub	657	0	15.82	7	24.36
SwaggerHub	2882	0	12.05	3	42.22
APIs.Guru	537	0	19.33	8	44.28
BigQuery	356	0	13.17	4	28.90
Combined	2882	0	14.94	7	29.52

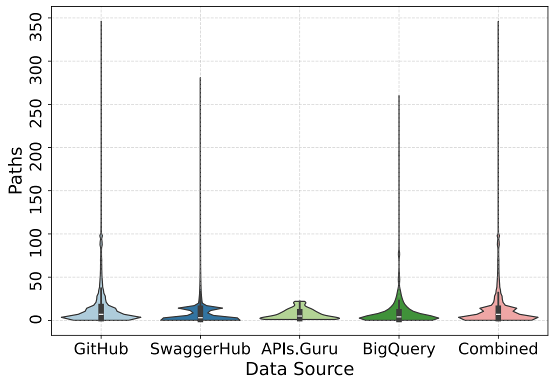


Figure 5: Comparative Analysis of Path Number Distributions Across All Sources

3.3 API Size: Structure and Datamodel

3.3.1 ***Are all the APIs across the datasets of the same structure size?*** Figure 5 showcases the path count distributions for the GitHub, SwaggerHub, BigQuery, and APIs.Guru datasets through violin plots. GitHub's data indicates a moderate skew with an average of 15.82 paths and a median of 7, pointing to some APIs with many more paths than others. SwaggerHub's distribution is more skewed, with an average path count of 12.05, a median of only 3, and a range extending to 2882, signaling that a few APIs possess a large number of paths. BigQuery's trend is akin to SwaggerHub, with an average of 13.17 and a median of 4, again showing a few APIs with elevated path counts. APIs.Guru differs, with a higher average path count of 19.33 and a median of 8, indicating a broader distribution of path counts among its APIs.

3.3.2 ***How does the size of datamodel vary across sources?*** As shown in Figure 6, for GitHub collection, the maximum number of schemas is 325, with an average of approximately 7.85, indicating a moderate concentration of APIs with a relatively small number of distinct schemas. SwaggerHub, however, shows a strikingly high maximum of 1479 distinct schemas, but its average is the lowest at around 0.35, suggesting that while most APIs have very few schemas, a few outliers are exceptionally having a complex datamodel. BigQuery's data displays a maximum of 286 schemas and an average of 10.31, aligning closely with GitHub's distribution. APIs.guru shows a higher diversity with a maximum of 360 schemas and an average of about 12.92, indicating a slightly wider range of complexity for datamodels in its APIs.

3.3.3 ***Do fresher APIs have larger structure?*** Analyzing the API paths in GitHub and Swagger datasets yearly, we can see from Figures 7 and 8 that the APIs exhibit a clear upward trend, indicative of an increasing expansion of APIs structures between 2014 and 2017. The intensity of this growth has decreased in the next

Dataset	Max	Min	Average	Median	StDev
GitHub	325	0	7.85	2	16.88
SwaggerHub	1149	0	0.35	0	2.96
APIs.Guru	360	0	12.92	4	28.64
BigQuery	286	0	10.54	4	26.26
Combined	1149	0	6.37	0	15.63

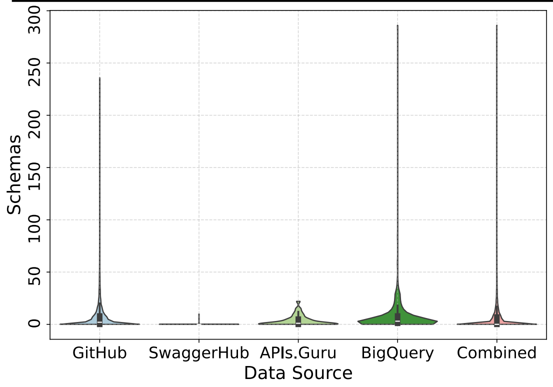


Figure 6: Comparative Analysis of Distinct Schema Number Distributions Across All Datasets

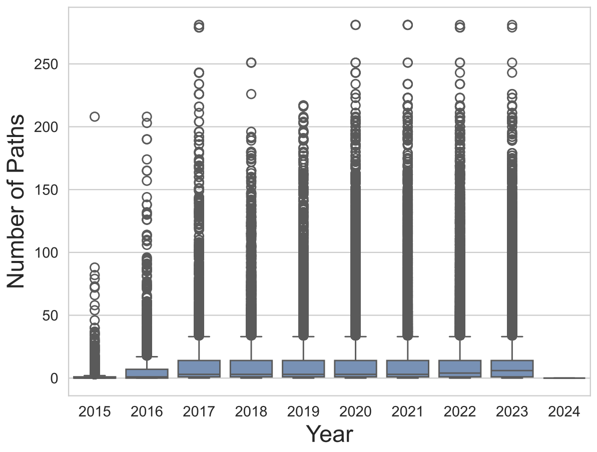


Figure 7: Distribution Number of Paths over the years in SwaggerHub Dataset

years. However, the SwaggerHub dataset shows a more pronounced variability and a higher presence of significant outliers, especially in the later years. But overall the structure size distributions did not exhibit major change over the years.

3.3.4 ***Do fresher APIs have larger datamodels?*** As shown in Figure 9, GitHub dataset reveals a consistent annual growth in the average number of distinct schemas within APIs. This shows that the specifications obtained from GitHub can be subject to a deeper Web API datamodel evolution analysis.

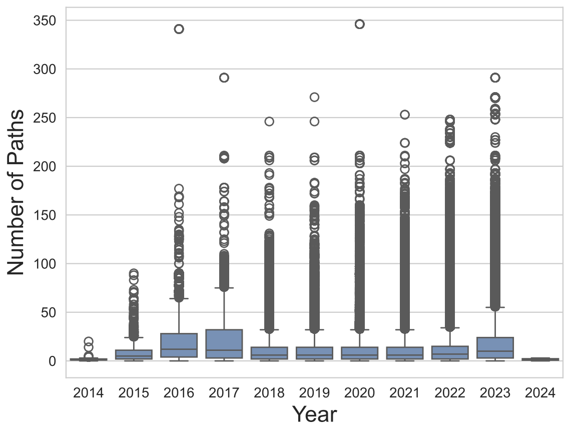


Figure 8: Distribution Number of Paths over the years in GitHub Dataset

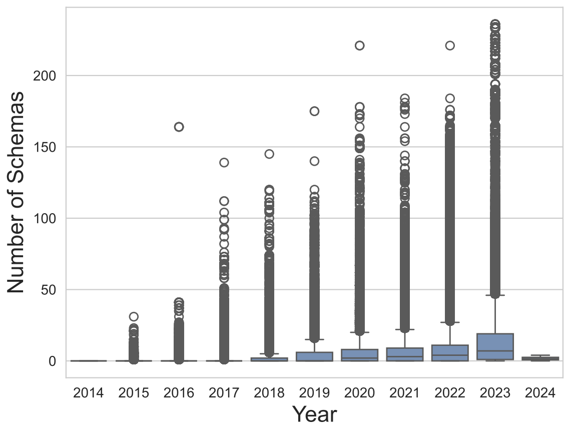


Figure 9: Distribution Number of Distinct Schemas over the years in GitHub

3.4 HTTP Methods Usage

3.4.1 ***Does the proportional usage of different HTTP methods change over time?*** Figures 11 and 10 show trends in API method usage over nine years, detailing the changing prevalence of methods like GET, POST, PUT. Each year is represented by a bar divided into sections for each method's count, with total operations annotated on top. Analysis of SwaggerHub and GitHub API operations reveals consistent proportions of HTTP method adoption across years. GET (read-only) remains most common, followed by POST (remote procedure calls). PUT and DELETE are used less frequently and do not show growth in recent years. Other methods like PATCH, HEAD, OPTIONS, and TRACE are rare.

3.5 API Maintenance Lifecycle

3.5.1 ***Were the API specifications just created and then abandoned?*** The Figure 12, illustrates the lifecycle of APIs in SwaggerHub descriptions, focusing on whether APIs created in a

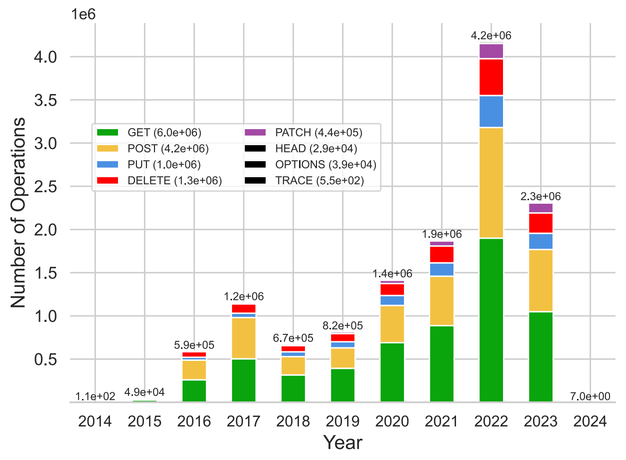


Figure 10: Yearly Trends in API Method Usage in GitHub Dataset

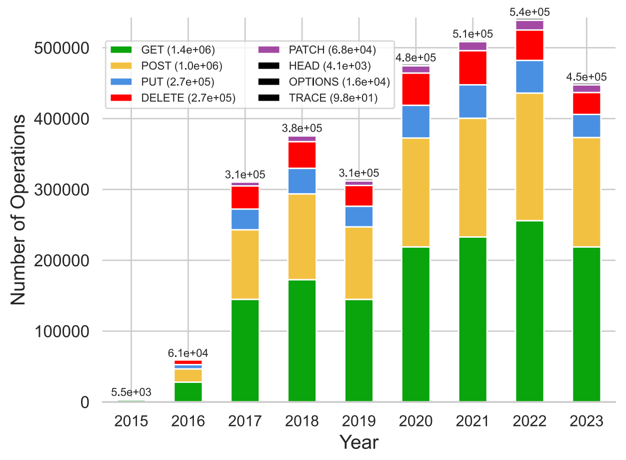
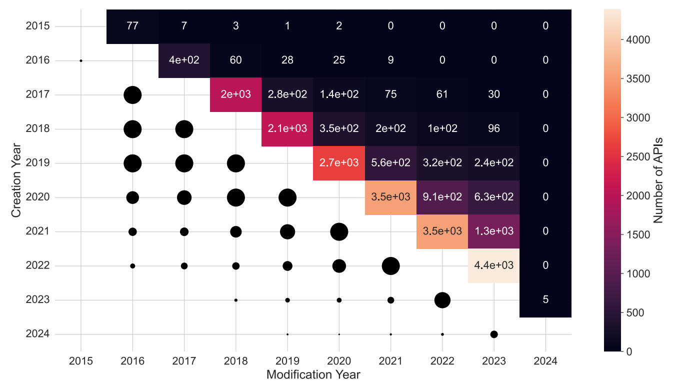


Figure 11: Yearly Trends in API Method Usage in SwaggerHub Dataset



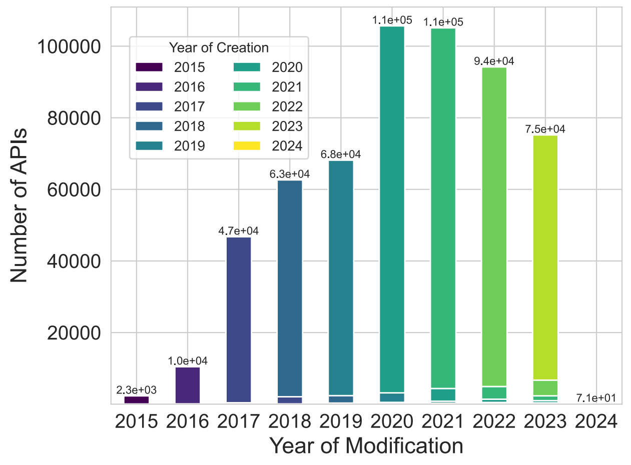


Figure 12: Number of Created and Modified APIs each year in the SwaggerHub Dataset

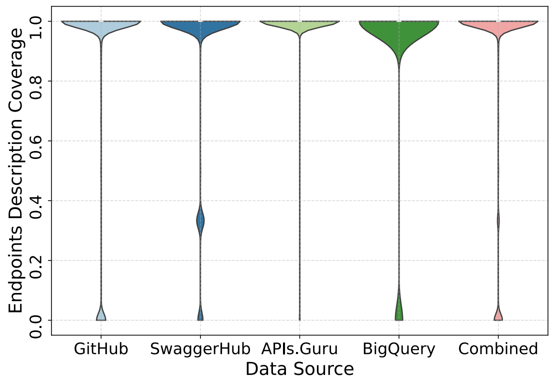


Figure 13: Comparative Analysis of Endpoint Description Coverage Distributions Across All Sources

specific year underwent modifications in subsequent years. It is important to note that initial creations are also counted as modifications. For instance, in 2023, Swagger Hub recorded 63,271 new API specifications. Additionally, there were 4,124 modifications to APIs originally created in 2022, 1,293 modifications to those from 2021, 620 from 2020, 242 from 2019, 96 from 2018, and 31 from 2017. Notably, APIs created in 2016 and 2015 have not been modified up to the present day.

3.6 Readability of Natural Language Documentation

3.6.1 ***How often are endpoints described in the APIs in each dataset?*** In Figure 13, all of the datasets show a concentration of values at the upper end, suggesting a significant number of endpoints with descriptions. SwaggerHub dataset displays a broader spread, indicating greater variability and a higher likelihood of described endpoints in and API.

3.6.2 ***How readable are the natural language operations descriptions found in OpenAPI specifications?*** For each data source, the distribution of the Coleman–Liau index (CLI) and Automated Readability Index (ARI) is illustrated in Figures 14 and 15.

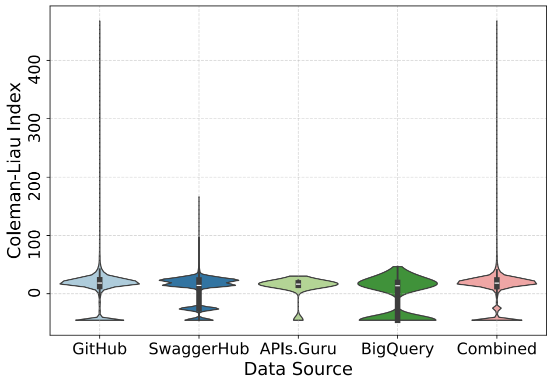


Figure 14: Comparative Analysis of Coleman-Liau Index Distributions Across Sources

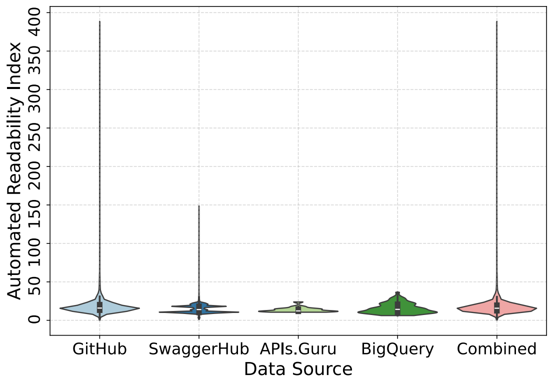


Figure 15: Comparative Analysis of the Automated Readability Index Distributions Across All Sources

These figures reveal relatively similar distributions for both indices, except for a few outliers in GitHub with exceptionally high values. This indicates that the complexity of text descriptions is consistent across datasets.

For instance, comparing two examples of from our dataset with distant index values, Elastic Email API, exhibits a high ARI of 22.78, suggesting a text complexity that demands an advanced understanding, likely targeting specialists or users with considerable expertise in the field. In contrast, the Cinema WebApp API, presents a significantly lower ARI of 9.49. This score typically characterizes APIs used for educational purposes or as examples, indicating a more accessible and user-friendly documentation, suitable for a broader audience, including beginners or students.

3.7 Types of Security Schemes in APIs

API security documentation is not required for a valid OpenAPI specification. The metrics we have established are not suitable for assessing the frequency of security mechanism adoption in API endpoints. Instead, they serve as a criterion to identify APIs with documented security features. Subsequent analyses can then be conducted on this filtered group of APIs.

Table 3: Number of APIs making use of different OpenAPI Security Schemes across different datasets

Type	GitHub	SwaggerHub	BigQuery	APIs.guru
apiKey	261358	6660	1770	528
oauth2	140227	5642	2651	2743
http	142906	2686	138	123
basic	37825	566	124	17
openIdConnect	4099	58	1	0
Secured APIs	26%	3%	74%	96%

Dataset	Max	Min	Average	Median	Std Dev
GitHub	1	0	0.47	0	0.49
SwaggerHub	1	0	0.03	0	0.15
APIs.Guru	1	0	0.69	1	0.46
BigQuery	1	0	0.51	1	0.49
Combined	1	0	0.30	0	0.45

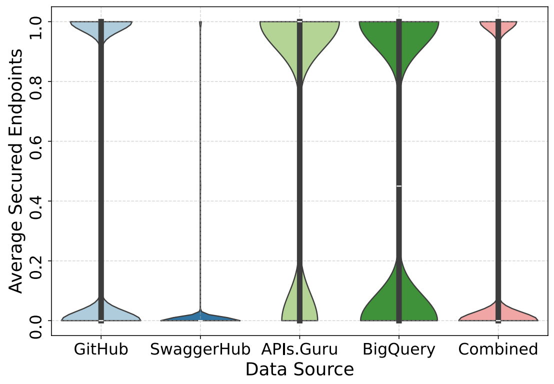


Figure 16: Comparative Analysis of Average Secured Endpoints Distributions Across All Sources

3.7.1 *How diverse are the security schemes types used in APIs?*

Table 3 reflects diverse security practices in APIs across GitHub, SwaggerHub, BigQuery, and APIs.guru. The usage of API keys is the most prevalent, followed by the OAuth2 protocol and the basic HTTP authentication scheme (described as 'http' in Swagger 2.0 and as 'basic' in the more recent OpenAPI 3.0). OpenIDConnect is present only in a small number of APIs.

We include in the table the security schemes that were used in at least three of the four source.

3.7.2 *What is the average number of secured endpoints in APIs across datasets?*

Github and APIs.Guru datasets show a higher average coverage of endpoints (Fig. 16), with GitHub being more consistent. SwaggerHub has notably lower coverage, while BigQuery dataset suggests a moderate coverage. Excluding SwaggerHub, we clearly observe a bi-modal distribution of artifacts where either there is a high level of coverage or no endpoints make use of security features. While almost all artifacts found in APIs.guru make use of security features, only very few of the API descriptions sourced from SwaggerHub do.

4 DISCUSSION AND USE CASES

The analysis finding reveals that the metrics which have been computed over the APIs included in the four datasets can be used to select and segment the API specifications for further analysis.

For example, APIs can be distinguished by their structure size (e.g., whether they have exactly one HTTP endpoint, such as SOAP or GraphQL APIs, or more such as RESTful APIs making use of URL addressing). Likewise, a rich set of API segments can be obtained by selecting specifications making use of a different set of HTTP method combinations (e.g., only GET for read-only, data-intensive APIs, or only POST for Remote Procedure style APIs).

Another set of metrics that can be used to further refine the sample selection concerns the API schemas, which tend to grow larger with more recent specifications. Static analysis studies to estimate the bandwidth costs of invoking or operating certain APIs can benefit from a collection of non-trivial API data models of a minimum level of complexity and size. These tend to be rare within the SwaggerHub dataset.

A key observation is the extensive coverage of human-readable descriptions across all sources, making the dataset valuable for natural language processing (NLP) tasks. Users of APIstic can leverage readability indexes to refine their selection, with higher indexes indicating descriptions likely authored by experts, as opposed to non-expert content like student projects or templates. A dataset rich in natural language endpoint descriptions is crucial for developing machine learning models for API design, as these models benefit from large datasets [55]. It provides context and semantics for improving generative models and supports tools for back-end/test code generation that use machine-readable API descriptions [23, 30, 39, 41]. Furthermore, this data can enhance the evaluation and testing of Web API documentation generation tools in terms of correctness and completeness [14].

Although OpenAPI does not mandate security documentation, the analysis found that security coverage varies widely, often being either completely absent or thoroughly documented with most of the paths covered by some security measure. This trend implies that documented security aspects are likely crucial in those APIs, making them candidate to be selected for further studies on API security documentation trends. This pattern is observed in the same way across all sources, with the exception of SwaggerHub, where most descriptions show minimal security coverage.

The dataset can also support empirical studies on the evolution of API specifications [63]. On the one hand it includes versioning metadata, which can help filter or compare stable API specifications vs. different types of preview releases [59]. On the other hand, many GitHub-sourced specifications are tracked throughout their history, which may include up to several hundred commits spanning across nine years [47], providing a way for researchers to find valuable case study subjects for longitudinal studies.

The provenance metadata attached to the metrics ensures the traceability of the exact source of each specification. In the case of Github and BigQuery it is often possible to retrieve the backend code in the same repository where the API description was found, which can allow other in-depth code analysis studies related to the implementation quality or performance benchmarking of the API [17], if it is possible to deploy and run it.

5 RELATED WORK

5.1 OpenAPI-Based Datasets

While many API hubs exist (Public APIs [1], RapidAPI Hub [2], APIs.guru, and SwaggerHub), only the last two offer OpenAPI descriptions and none are known to provide detailed API analytics metrics.

Assetnote, an Australian information security company, conducted in 2021 an analysis using a dataset of OpenAPI specifications for their contextual content discovery project [4]. Our data collection process shares some of their sources (such as BigQuery, APIs.guru and SwaggerHub) but has been improved. For example, looking on BigQuery for files named "swagger.json", "openapi.json", and "api-docs.json" returned 11,000 files, overlooking API specifications with non-standard file names. In contrast, our query (Listing 1) obtained 33,957 API specifications, resulting in 6,376 valid ones for our dataset. Our SwaggerHub crawler goes beyond the default limit of 10,000 results per request. Given the amount of API specifications we managed to retrieve so far by mining software repositories we have not yet attempted to reproduce their “Scanning the Internet” strategy, which resulted in about 44,000 API specifications, retrieved by attempting to GET 22 predefined URL templates during a massive internet-wide scan.

Other research initiatives have constructed specialized datasets and metrics for APIs, analysing OpenAPI specifications, such as [68], where Yaghoub-Zadeh-Fard and Benatallah present the API2CAN dataset and service, designed to aid in building natural language interfaces for REST APIs. The dataset, featuring 14,370 API method and user utterance pairs from 983 APIs in APIs.guru, targets the development of efficient chatbots. Martin-Lopez et al. used a dataset of 40 real-world web APIs (2,557 operations) to examine the role of inter-parameter dependencies in Web APIs, where the found a lack of formal representation for these dependencies in API design languages like OpenAPI [50].

5.2 Web API Metrics

In [19], the authors propose RAMA, a prototypical tool which automatically computes ten maintainability metrics for service-based systems from machine-readable API descriptions in different formats (OpenAPI/Swagger, RAML [9], WADL [35]). The collected structural metrics describe mainly the characteristics of the: paths (e.g, Path Length, longest path), operations (e.g: arguments per path). The metrics were based on Haupt et al.’s study [36]. While our metrics are all oriented to measure API components sizes, Haupt et al. defined seven structural metrics for the API tree structure and the type of operations it provides: max depth,#resources, #read only resources, # links, #DELETE, #POST #roots. These metrics have been applied to 286 real-world API described in OpenAPI/Swagger.

Additional metrics implemented in RAMA focus on the relationships between the operations inputs/outputs (e.g: lack of message-level cohesion [16], service interface data cohesion [54]). We plan to integrate RAMA’s metrics as a future extension of the APIstic dataset to give a more complete reflection on the relationship between operations and the corresponding message payloads and a more detailed view on the API structure.

Inzunza et al. propose methods to identify missing or incomplete elements in API documentation in their study [38]. Cheh and Chen

Table 4: Summary of Studies on Web Service APIs

Year	# of Artifacts/People	Source of Data	Quality Attributes/Metrics	Study Topic
[31] 2014	APIs from 4 companies	Twitter, Google Maps, Facebook, Netflix	Evolution challenges	Challenges in API evolution
[52] 2018	500 APIs	Alexa's top 4000 sites	REST principles, technical features	API features and REST compliance
[69] 2018	4 API types	Uber, WordPress, OpenStack, Media Processing	Learnability, stability	Propose a quality model for Web APIs
[67] 2019	10 developers	Google Auth API	Usability issues	Usability issues in APIs
[17] 2020	3-month benchmark	Web APIs (various)	Availability, performance, security	Analyze and compare web API qualities
[33] 2020	9,714 URLs across 3,376 apps	Android apps	Security aspects	Web communication security in mobile apps
[43] 2020	Usage logs	DHIS2 Web API at WHO	Usability attributes (clarity, consistency, etc.)	Assess web API usability
[73] 2021	2,000 APIs	APIs.guru	REST API design guidelines	REST API design quality
[27] 2021	2 000 APIs, 10 505 endpoints	APIs.guru	Natural Language Processing	Topic Extraction
[71] 2022	20,047 APIs, 1,885 questions	ProgrammableWeb, APIs.guru, Stack Overflow	User issues, features	Common user issues and expectations
[20] 2023	12 API snippets	-	RESTful design rules	Impact of RESTful rules on understandability

introduce a semi-automatic approach for detecting security issues in API specifications in their work [24]. They focus on identifying sensitive and insensitive data fields, insecure or high-risk API calls that might leak sensitive data, and calculating the exposure level of each data field and API call.

5.3 Web API Landscape Analytics

The landscape of Web APIs has been extensively explored with studies focusing on diverse aspects such as compliance to REST design principles [66], usability and learnability, stability and evolution, security and performance (Table 4).

Technical Features and REST Principles. Neumann's study [52] examines the technical aspects and REST compliance of 500 web service APIs, noting widespread JSON use and auto-generated documentation but low REST adherence, calling for more standardization. Complementing this, Zhou et al.'s work [73] assesses REST API design quality of the specifications of APIs.guru.

Usability and Design Rules. Web API usability is a key focus in several studies. Koçi et al.'s research [43] adopts a data-driven approach to evaluate web API usability, underscoring the importance of attributes such as clarity and consistency. This resonates with Wijayarathna and Arachchilage's examination of the Google Authentication API [67], which points out significant usability concerns likes inadequate documentation and ambiguous error messaging. Likewise, Bogner et al. highlights the relationship between RESTful API design principles, understandability and consistency[20].

API Evolution and Client Developer Challenges. In [31], Espinha et al. study addresses the challenges faced by client developers due to the evolution of web APIs. The frequent and unpredictable changes in APIs like those of Twitter and Google Maps highlight the significant maintenance efforts required from developers and the lack of industry standards in API evolution policies.

API Usage and Consumer Perspective. In [69], Yamamoto et al. proposes a quality model for REST-based Web APIs focusing on learnability and stability, two qualities vital from a consumer's perspective. This model provides metrics that are validated empirically, offering a framework to assess and improve the usability and stability of Web APIs. In contrast, Zhang et al.'s study of 20,047 web APIs focuses on common user issues and expectations, providing insights for developers and registry managers to enhance the functionality and management of web APIs [71].

Security and Performance. Gadient et al. discuss the security aspects of web communication in mobile apps in [33]. Their research emphasizes the need for secure protocols in web API design due to prevalent insecure HTTP connections and security flaws in client-server communications. In [17], Bermbach and Wittern examine the runtime quality of web APIs during 2015 and 2018, focusing on variations in availability, latency, and security. The study finds significant regional and temporal differences in API performance and responsiveness.

6 CONCLUSIONS

This paper contributes to the growing body of knowledge in API analytics by introducing a unique and novel dataset that captures a diverse range of API characteristics.

This dataset serves as a valuable resource to extract samples for large-scale empirical studies in a field where most existing studies have been performed considering up to a few thousands of artifacts. The current version of the dataset, presented in this paper, comprises measurements and provenance metadata of a substantial number of 1,275,568 API descriptions sourced from GitHub, SwaggerHub, BigQuery and APIs.guru, providing a comprehensive source to analyze current API practices and trends in API design, documentation, security strategies, and data modeling. Particularly noteworthy is the portion of the dataset sourced from GitHub, which includes historical data of APIs, providing insights into the evolution of these metrics over time.

We are dedicated to the ongoing enrichment of this dataset, continually updating it with newly fetched and validated API specifications to ensure it remains a relevant and comprehensive resource for the study and application of API analytics.

For future work, to encourage good coding practices such as those based on Richardson's REST maturity model [66, 73], we intend to introduce more metrics and indicators to aid in further API classifications (e.g., whether they are meant for public [52] or private consumption within a microservice architecture). Likewise, we are investigating how to aggregate fine-grained security metrics (e.g. [24]) so that APIs with a high level of exposure can be detected [34]. We are also considering the possibility to let users compute their own custom metrics by submitting JSON queries to be applied on every API specification.

Acknowledgements. This work was supported by the SNF with the API-ACE project 184692.

REFERENCES

[1] Public APIs — A directory of free and public apis. https://publicapis.io/.
[2] Rapid API Hub. https://rapidapi.com/hub.
[3] APIs.Guru. https://github.com/APIs-guru/openAPI-directory https://APIs.guru/.
[4] Contextual Content Discovery: You've forgotten about the API endpoints. https://blog.assetnote.io/2021/04/05/contextual-content-discovery/.
[5] Google BigQuery API. https://cloud.google.com/bigquery/docs/reference/rest/.
[6] GitHub API. https://docs.github.com/en/rest, .
[7] GitHub Archive. http://www.gharchive.org/, .
[8] OpenAPI Initiative. https://www.openAPIs.org/.
[9] RAML. https://raml.org/. Accessed: 2021-06-01.
[10] Semantic Versioning. https://semver.org/.
[11] SwaggerHub. https://swagger.io/tools/swaggerhub/, .
[12] SwaggerHub Registry API. https://app.swaggerhub.com/APIs-docs/swagger-hub/registry-API/, .
[13] Swagger Validator. https://github.com/APIDevTools/swagger-parser.
[14] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. Software documentation issues unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1199–1210. IEEE, 2019.
[15] Tarfah Alrashed, Jumana Almahmoud, Amy X Zhang, and David R Karger. Scrapir: making web data apis accessible to end users. In *Proceedings of the 2020 CHI conference on human factors in computing systems*, pages 1–12, 2020.
[16] Dionysis Athanasopoulos, Apostolos V Zarras, George Miskos, Valerie Issarny, and Panos Vassiliadis. Cohesion-driven decomposition of service interfaces without access to source code. *IEEE Transactions on Services Computing*, 8(4): 550–562, 2014.
[17] David Bermbach and Erik Wittern. Benchmarking web api quality-revisited. *Journal of Web Engineering*, 19(5-6):603–646, 2020.
[18] Matthias Biehl. *Webhooks–Events for RESTful APIs*, volume 4. API-University Press, 2017.
[19] Justus Bogner, Stefan Wagner, and Alfred Zimmermann. Collecting service-based maintainability metrics from restful api descriptions: static analysis and threshold derivation. In *European Conference on Software Architecture*, pages 215–227. Springer, 2020. doi: http://dx.doi.org/10.1007/978-3-030-59155-7_16.
[20] Justus Bogner, Sebastian Kotstein, and Timo Pfaff. Do restful api design rules have an impact on the understandability of web apis? *Empirical Software Engineering*, 28(6):1–35, 2023. doi: http://dx.doi.org/10.1007/s10664-023-10367-y.
[21] Pierre Bourhis, Juan L Reutter, Fernando Suárez, and Domagoj Vrgoč. Json: data model, query languages and schema specification. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems*, pages 123–135, 2017. doi: https://doi.org/10.1145/3034786.3056120.
[22] Engin Bozdag, Ali Mesbah, and Arie van Deursen. A comparison of push and pull techniques for ajax. In *Proceedings of the 2007 9th IEEE International Workshop on Web Site Evolution*, WSE '07, page 15–22, 2007. ISBN 9781424414505. doi: 10.1109/WSE.2007.4380239.
[23] Steven Bucaille, Javier Luis Cánovas Izquierdo, Hamza Ed-Douibi, and Jordi Cabot. An openapi-based testing framework to monitor non-functional properties of rest apis. In *International Conference on Web Engineering*, pages 533–537. Springer, 2020.
[24] Carmen Cheh and Binbin Chen. Analyzing openapi specifications for security design issues. In *2021 IEEE Secure Development Conference (SecDev)*, pages 15–22. IEEE, 2021. doi: 10.1109/SecDev51306.2021.00019.
[25] Hsiao-Jung Chen, Shang-Pin Ma, and Hsueh-Cheng Lu. Collaborative security annotation and online testing for web apis. In *2021 IEEE International Conference on e-Business Engineering (ICEBE)*, pages 9–15. IEEE, 2021.
[26] Meri Coleman and Ta Lin Liau. A computer readability formula designed for machine scoring. *Journal of Applied Psychology*, 60(2):283, 1975.
[27] Leonardo da Rocha Araujo, Guillermo Rodríguez, Santiago Vidal, Claudia Marcos, and Rodrigo Pereira dos Santos. Empirical analysis on openapi topic exploration and discovery to support the developer community. *Computing and Informatics*, 40(6):1345–1369, 2021.
[28] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. Sampling projects in github for msr studies. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 560–564. IEEE, 2021.
[29] Alan De Renzis, Martin Garriga, Andres Flores, Alejandra Cechich, Cristian Mateos, and Alejandro Zunino. A domain independent readability metric for web service descriptions. *Computer Standards & Interfaces*, 50:124–141, 2017.
[30] Hamza Ed-Douibi, Javier Luis Cánovas Izquierdo, and Jordi Cabot. Automatic generation of test cases for rest apis: A specification-based approach. In *2018 IEEE 22nd international enterprise distributed object computing conference (EDOC)*, pages 181–190. IEEE, 2018.
[31] Tiago Espinha, Andy Zaidman, and Hans-Gerhard Gross. Web api growing pains: Stories from client developers and their code. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 84–93, 2014. doi: https://doi.org/10.1109/CSMR-WCRE.2014.6747228.
[32] Sérgio Fernandes and Jorge Bernardino. What is bigquery? In *Proceedings of the 19th International Database Engineering & Applications Symposium*, pages 202–203, 2015.
[33] Pascal Gadient, Mohammad Ghafari, Marc-Andrea Tarnutzer, and Oscar Nierstrasz. Web apis in android through the lens of security. In *2020 IEEE 27th international conference on software analysis, evolution and reengineering (SANER)*, pages 13–22. IEEE, 2020.
[34] Patric Genfer and Uwe Zdun. Avoiding excessive data exposure through microservice apis. In *European Conference on Software Architecture*, pages 3–18. Springer, 2022.
[35] Marc J. Hadley. Web application description language (wadl). Technical report, USA, 2006.
[36] Florian Haupt, Frank Leymann, Anton Scherer, and Karolina Vukojevic-Haupt. A framework for the structural analysis of rest apis. In *Proc. IEEE International Conference on Software Architecture (ICSA)*, pages 55–58, 2017.
[37] J. Higginbotham. *Principles of Web API Design: Delivering Value with APIs and Microservices*. Addison-Wesley Signature Series. Pearson Education (US), 2021.
[38] Sergio Inzunza, Reyes Juárez-Ramírez, and Samantha Jiménez. Api documentation: A conceptual evaluation model. In *Proc. of World Conference on Information Systems and Technologies (WorkdCIST'18): Trends and Advances in Information Systems and Technologies*, pages 229–239. Springer, 2018. doi: 10.1007/978-3-319-77712-2_22.
[39] Stefan Karlsson, Adnan Čaušević, and Daniel Sundmark. Quickrest: Property-based test generation of openapi-described restful apis. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 131–141. IEEE, 2020.
[40] Rick Kazman, Roberto Tonelli, and Cesare Pautasso. An empirical basis for software architecture research. In *Software Architecture: Research Roadmaps from the Community*, pages 87–100. Springer, 2023.
[41] Myeongsoo Kim, Davide Corradini, Saurabh Sinha, Alessandro Orso, Michele Pasqua, Rachel Tzoref-Brill, and Mariano Ceccato. Enhancing rest api testing with nlp techniques. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1232–1243, 2023.
[42] Myeongsoo Kim, Saurabh Sinha, and Alessandro Orso. Adaptive rest api testing with reinforcement learning. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 446–458. IEEE, 2023.
[43] Rediana Koçi, Xavier Franch, Petar Jovanovic, and Alberto Abelló. A data-driven approach to measure the usability of web apis. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 64–71. IEEE, 2020. doi: https://doi.org/10.1109/SEAA51224.2020.00021.
[44] Rediana Koçi, Xavier Franch, Petar Jovanovic, and Alberto Abelló. Web api evolution patterns: A usage-driven approach. *Journal of Systems and Software*, 198:111609, 2023. doi: 10.1016/j.jss.2023.111609.
[45] Valliappa Lakshmanan and Jordan Tigani. *Google Bigquery: the definitive guide: data warehousing, analytics, and machine learning at scale*. O'Reilly Media, 2019.
[46] Arnaud Lauret. *The design of web APIs*. Simon and Schuster, 2019.
[47] Fabio Di Lauro, Souhaila Serbout, and Cesare Pautasso. A large-scale empirical assessment of web api size evolution. *Journal of Web Engineering*, 21:1937–1980, November 2022. doi: 10.13052/jwe1540-9589.2167.
[48] Daniel Lübke, Olaf Zimmermann, Cesare Pautasso, Uwe Zdun, and Mirko Stocker. Interface evolution patterns: Balancing compatibility and extensibility across service life cycles. In *Proceedings of the 24th European Conference on Pattern Languages of Programs (EuroPLoP)*, pages 1–24, 2019.
[49] Shang-Pin Ma, Ming-Jen Hsu, Hsiao-Jung Chen, and Yu-Sheng Su. Api prober–a tool for analyzing web api features and clustering web apis. *ICEBE 2019: Advances in E-Business Engineering for Ubiquitous Computing*, 2019.
[50] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. Inter-parameter dependencies in real-world web apis: The idea dataset. In *Research and Evidence in Software Engineering*, pages 101–106. Auerbach Publications, 2021.
[51] Mehdi Medjaoui, Erik Wilde, Ronnie Mitra, and Mike Amundsen. *Continuous API management*. O'Reilly, 2021.
[52] Andy Neumann, Nuno Laranjeiro, and Jorge Bernardino. An analysis of public rest web service apis. *IEEE Transactions on Services Computing*, 14(4):957–970, 2018. doi: https://doi.org/10.1109/TSC.2018.2847344.
[53] Chris Parnin and Christoph Treude. Measuring api documentation on the web. In *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*, pages 25–30, 2011.
[54] Mikhail Perepletchikov, Caspar Ryan, and Keith Frampton. Cohesion metrics for predicting maintainability of service-oriented software. In *Seventh International Conference on Quality Software (QSIC 2007)*, pages 328–335. IEEE, 2007.
[55] Julian Aron Prenner and Romain Robbes. Making the most of small software engineering datasets with modern machine learning. *IEEE Transactions on Software Engineering*, 48(12):5050–5067, 2021.
[56] Yuanbo Qiu. The openness of open application programming interfaces. *Information, Communication & Society*, 20(11):1720–1736, 2017.
[57] Carlos Rodríguez, Marcos Baez, Florian Daniel, Fabio Casati, Juan Carlos Trabucco, Luigi Canali, and Gianraffaele Percannella. Rest apis: A large-scale analysis

of compliance with principles and best practices. In *Proc. 16th International Conference on Web Engineering (ICWE)*, pages 21–39. Springer, 2016.

[58] Jéssica S Santos, Leonardo G Azevedo, Elton Soares, Raphael Thiago, and Viviane T Silva. Analysis of tools for rest contract specification in swagger/openapi. In *International Conference on Enterprise Information Systems*. SciTePress, 2020.

[59] Souhaila Serbout and Cesare Pautasso. An empirical study of web api versioning practices. In *Proc. 23rd International Conference on Web Engineering (ICWE)*. Springer, 2023. ISBN 978-3-031-34443-5. doi: 10.1007/978-3-031-34444-2_22. URL https://doi.org/10.1007/978-3-031-34444-2_22.

[60] Souhaila Serbout, Fabio Di Lauro, and Cesare Pautasso. Web apis structures and data models analysis. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, pages 84–91. IEEE, 2022.

[61] Souhaila Serbout, Amine El Malki, Cesare Pautasso, and Uwe Zdun. Api rate limit adoption - a pattern collection. In *Proc. 28th European Conference on Pattern Languages of Programs (EuroPLoP 2023)*, Kloster Irsee, Germany, July 2023. ACM, ACM.

[62] Edgar A Smith and RJ Senter. *Automated readability index*. Number AD0667273. Aerospace Medical Research Laboratories, November 1967. https://apps.dtic.mil/sti/citations/AD0667273.

[63] Mirko Stocker and Olaf Zimmermann. Api refactoring to patterns - catalog, template and tools for remote interface evolution. In *Proc. 28th European Conference on Pattern Languages of Programs (EuroPLoP)*, 2023.

[64] Mirko Stocker, Olaf Zimmermann, Uwe Zdun, Daniel Lübke, and Cesare Pautasso. Interface quality patterns: Communicating and improving the quality of microservices apis. In *Proceedings of the 23rd European conference on pattern languages of programs*, pages 1–16, 2018.

[65] Christoph Treude, Justin Middleton, and Thushari 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*, ESEC/FSE, page 1509–1512, 2020. ISBN 9781450370431. doi: 10.1145/3368089.3417045.

[66] Jim Webber, Savas Parastatidis, and Ian Robinson. *REST in practice: Hypermedia and systems architecture*. O'Reilly, 2010.

[67] Chamila Wijayarathna and Nalin AG Arachchilage. An empirical usability analysis of the google authentication apis. In *Proceedings of the 23rd International Conference on Evaluation and Assessment in Software Engineering*, pages 268–274, 2019. doi: https://doi.org/10.1145/3319008.3319350.

[68] Mohammad-Ali Yaghoub-Zadeh-Fard and Boualem Benatallah. Api2can: a dataset & service for canonical utterance generation for rest apis. *BMC Research Notes*, 14:1–3, 2021.

[69] Rieko Yamamoto, Kyoko Ohashi, Masahiro Fukuyori, Kosaku Kimura, Atsuji Sekiguchi, Ryuichi Umekawa, Tadahiro Uehara, and Mikio Aoyama. A quality model and its quantitative evaluation method for web apis. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 598–607. IEEE, 2018.

[70] Uwe Zdun, Mirko Stocker, Olaf Zimmermann, Cesare Pautasso, and Daniel Lübke. Guiding architectural decision making on quality aspects of microservice apis. In *Proc. 16th International Conference on Service-Oriented Computing (ICSOC 2018)*, volume 11236, pages 73–89, Hangzhou, Zhejiang, China, November 2018. Springer, Springer. doi: 10.1007/978-3-030-03596-9_5.

[71] Neng Zhang, Ying Zou, Xin Xia, Qiao Huang, David Lo, and Shanping Li. Web apis: Features, issues, and expectations–a large-scale empirical study of web apis from two publicly accessible registries using stack overflow and a user survey. *IEEE Transactions on Software Engineering*, 49(2):498–528, 2022. doi: https://doi.org/10.1109/TSE.2022.3154769.

[72] Shixiang Zhou, Heejin Jeong, and Paul A Green. How consistent are the best-known readability equations in estimating the readability of design standards? *IEEE Transactions on Professional Communication*, 60(1):97–111, 2017.

[73] Xin-Yu Zhou, Wei Chen, Guo-Quan Wu, and Wei Jun. Rest api design analysis and empirical study. *Journal of Software*, 33(9):3271–3296, 2021.

[74] Olaf Zimmermann, Mirko Stocker, Daniel Lubke, Uwe Zdun, and Cesare Pautasso. *Patterns for API design: simplifying integration with loosely coupled message exchanges*. Addison-Wesley Professional, 2022.