

Master Thesis

---

# Exploring REST Design Practices in ML Web APIs: A Qualitative Approach

---

by

**Roger Ruiz Salvat**

(2783192)

*First supervisor:* Justus Bogner  
*Daily supervisor:* Justus Bogner  
*Second reader:* Filip Ilievski

August 8, 2024

*Submitted in partial fulfillment of the requirements for  
the VU degree of Master of Science in Information Sciences*

## DECLARATION OF AUTHORSHIP

I, **Roger Ruiz Salvat**, declare that this thesis titled, **Exploring REST Design Practices in ML Web APIs: A Qualitative Approach** and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Roger

Date: 08-08-2024

# Exploring REST Design Practices in ML Web APIs: A Qualitative Approach

Roger Ruiz Salvat

Vrije Universiteit Amsterdam

Amsterdam, The Netherlands

[r.ruiz.salvat@student.vu.nl](mailto:r.ruiz.salvat@student.vu.nl)

## ABSTRACT

Machine learning (ML) is becoming increasingly important for organizations to perform analysis and make predictions based on data. Consequently, cloud computing platforms are integrating ML functionalities and services. This is because cloud infrastructures are a suitable option for performing ML operations that require high computing power, distributed computing, and data storage.

To access and manage ML resources, Machine Learning as a Service (MLaaS) platforms offer various interfaces. In this study, we analyze common design practices in the design of ML cloud web Application Programming Interfaces (APIs) and their adherence to representational state transfer (REST) design rules. Five APIs were selected for the study: Amazon SageMaker, Azure ML, Google Cloud Vertex AI, IBM ML, and the BigML API.

This research aims to explore how the main ML cloud providers design their APIs and assess the extent to which they conform to REST rules. Obtaining then a comparison between the available ML APIs in terms of understandability and usability. The outcome of the analysis provides insights for developers who need to implement ML cloud solutions.

To explore common design practices among MLaaS APIs, we employed thematic analysis, a qualitative method for theory development, on online API documentation. We then investigated the adherence of the studied APIs to REST rules by manually analysing API endpoints and examining the API documentation.

The study found that API versioning is an important factor and all APIs consider it. Additionally, all APIs except Azure ML include links to data entities in their response payloads. However, using hypermedia as the state engine (HATEOAS) seems to be less common among the studied APIs. Enhancing caching in responses was also found to be of lesser importance. The handling of resource updates remains unclear, as many different approaches were detected. Furthermore, APIs handle unauthorized access uniformly, but vary in responses to forbidden access and malformed requests.

Lastly, all MLaaS platforms seem to adhere to most of the studied REST design rules, with an average compliance rate of 74.19%. However, for Amazon SageMaker, most rules remain undetermined due to its approach of wrapping web APIs into Software Development Kits (SDKs).

## 1 INTRODUCTION

Artificial Intelligence (AI) is a broad and diverse field that encompasses various subjects such as indexing algorithms, reasoning, learning models, natural language processing, speech-recognition, vision systems, etc. In contrast, ML is a specific approach within AI that aims to achieve intelligent behavior through training and learning processes. The core concept of ML is statistical learning

theory, employing statistical methods to enable computers to learn from data and make predictions or decisions [14]. ML has become a key technology for Industry 4.0, which refers to the continuous automation of traditional manufacturing and industrial processes using advanced technologies. Its applications span across various domains such as cybersecurity, healthcare, e-commerce, and agriculture. ML encompasses a range of techniques including classification tasks, regressions, clustering, and reinforcement learning [25].

An ML model can be defined as a computer program specifically designed to make predictions by leveraging statistical methods and analyzing data. Training and building an ML model might require specialized hardware, such as GPUs, as some ML processes are computationally expensive. Moreover, ML developers may want to have the models accessible at any time. Therefore, using cloud infrastructures can facilitate many tasks needed for deploying an ML model. Then, developers can also reduce costs and manage ML resources efficiently [8]. There exist three main types of cloud computing solutions:

- **Software as a Service (SaaS):** This allows consumers to access applications provided by the cloud provider, hosted on a cloud infrastructure. These applications can be accessed from various devices using a client interface.
- **Infrastructure as a Service (IaaS):** In contrast, IaaS offers full control over processing and storage. This means consumers can deploy and manage their own software.
- **Platform as a Service (PaaS):** Finally, PaaS allows consumers to leverage the cloud infrastructure for deploying their applications on pre-configured environments. The cloud provider takes responsibility for controlling and managing the underlying physical and virtual infrastructure, including networks, servers, operating systems, and storage [1].

For ML solutions, computationally intensive tasks like processing datasets and training deep neural networks are common. Machine Learning as a Service (MLaaS), a type of PaaS, allows users to upload their data and models to the cloud for training. In addition, MLaaS can serve as a platform for deploying models and using them to make predictions (performing inference tasks) within the cloud environment [23].

MLaaS platforms provide different interfaces for users to interact with the platform: programming-oriented interfaces, comprehensive user interfaces, or a combination of both. These interfaces, including application programming interfaces (APIs), allow other systems to communicate with the cloud infrastructure to use or

manage ML resources. Web APIs specifically use HyperText Transfer Protocol (HTTP) methods to communicate with client applications. To facilitate these client-server communications, ML platforms implement representational state transfer (REST) software architectures.

Identifying common design practices can reveal the level of interoperability and understandability among ML APIs. Potentially uncovering existing API security vulnerabilities. This research aims to outline design strategies for optimizing resource utilization, minimizing latency, and handling substantial workloads. Subsequently, by considering emerging technologies and user needs, future work could develop guidelines for integrating ML services into various environments. Despite the popularity that REST architectures have been acquiring during the past years [12], some studies suggest that web APIs often fail to adhere to all the design guidelines and therefore they cannot be considered as RESTful APIs [2][24][19]. In this study, we also aim to assess whether MLaaS APIs conform to REST design rules.

Five MLaaS APIs have been selected for this study. All APIs offer the capability to train ML models and are maintained by different cloud providers. The research has been approached into the following research questions:

RQ1 What are the existing practices<sup>1</sup> for designing and specifying web APIs for ML cloud services?

RQ2 How closely do ML APIs adhere to REST API design rules?

Although some existing research explores REST design practices in web APIs [6][15], we aim to make a distinct contribution by specifically studying web APIs within MLaaS platforms. Since ML is a rapidly evolving field within software engineering, our goal is to investigate how this technology influences the design of web APIs in cloud infrastructures. Furthermore, we aim to analyze how well MLaaS services are implementing existing REST design practices.

Section 2 offers a contextualization of the project scope and discusses related works that contribute to the broader study of REST APIs. Moving forward, in section 3, we delve into the methodology implemented to address the research questions at hand. Following that, in section 4, we present the results obtained from the qualitative study conducted. Section 5 discusses findings that emerged during the study and also highlight potential avenues for future research. We then shift our focus to section 6, where we thoroughly examine the limitations and threats to the validity of our study. Lastly, in section 7, we present the conclusions drawn from our study, wrapping up the paper with key takeaways.

## 1.1 Personal contributions

This paper is part of a collaborative effort to study web API design practices for two types of APIs:

- APIs to perform ML operations, as introduced in this paper.
- APIs to use generative AI (GenAI) functionalities.

The research methodology is the same for both studies as they aim to answer the same research questions. Similarly, the background and related work on the study of REST design practices is the same for both analyses, as our contribution focuses on applying

the analysis to ML/GenAI APIs. However, the study objects differ, and consequently, the results obtained and the conclusions derived also differ.

## 2 BACKGROUND AND RELATED WORK

On designing web APIs, Massé [16] compiled an extensive set of Representational State Transfer (REST) rules and best practices for designing RESTful web services. The aim was to establish standards and promote consistency among web APIs to enhance reusability and improve the maintainability of web services. The set of proposed rules encompass several key aspects of web API design, including HTTP methods, URI design, metadata specifications, and error response handling. Violating these REST rules can significantly hinder the understandability of APIs. This correlation underscores the importance of adhering to design principles to enhance both the maintainability and usability of APIs [4].

Depending on the extent to which those design rules and best practices are followed, Richardson [10] proposed a maturity model for web APIs. Higher levels of maturity indicate that the APIs adhere to more of the best practices in REST, with the use of hypermedia being a requirement for achieving the highest level of maturity.

Kotstein et al. [15] present a Delphi study that assessed the importance of the rules proposed by Massé [16]. The study found that 28 out of the examined rules were highly important, while 17 were categorized as having medium importance. These rules primarily aim to achieve level 2 of the Richardson’s Maturity Model, focusing on HTTP mechanisms and semantics. However, rules related to level 3 (hypermedia) were considered less critical. Additionally, the study revealed that the highly important rules were mainly associated with quality attributes like usability, maintainability, and compatibility. Conversely, they showed a weaker correlation with performance efficiency, reliability, and security.

On the study of MLaaS platforms, Kaymakci et al. [13] analyze the adoption of ML for small and medium-sized enterprises (SMEs). They identify criteria to evaluate which ML cloud API solution is most suitable for a specific SME. The criteria points are organized into categories derived from quality requirements such as security, reliability, performance, and costs. The analysis of the study objects differs from our research as the authors followed a design science research strategy. The end result is a framework that developers can use to make informed decisions. The framework was validated through a case study where the Google Cloud, Azure, and AWS platforms were analyzed for their suitability for a German manufacturing company. In that sense, the study uses three study objects that were also considered in our study.

On ML APIs, Cetinsoy et al. [5] discuss the evolution of ML APIs and some of their popular algorithms. Additionally, they present current challenges such as the time value of fraud detection models (real-time inference) or the data complexity of datasets used to build recommender systems. The authors claim that providing ML functionalities through a cloud REST API, where each ML element is a resource, could help address these challenges. Furthermore, such an approach facilitates users in building fully automated ML workflows. Although Cetinsoy et al. [5] provide a review of the state of ML APIs rather than conducting a qualitative research

<sup>1</sup>Practices in our interpretation are a combination of response messages, metadata specifications, and interaction design, along with existing similarities and differences among them.

study, they are writing from the perspective of BigML, one of the platforms considered in our research.

Coblenz et al. [6] conducted interviews to identify common best practices in web API design. The study observed that certain practices, such as Hypermedia as the Engine of Application State (HATEOAS), are not typically implemented. Additionally, the interviews revealed that implementing authorization and authentication protocols and ensuring client-API compatibility were the most challenging aspects of API design. Furthermore, the analysis identified a tendency to misuse HTTP verbs and found that 5xx response errors often lacked sufficient information for effective debugging. The approach followed by Coblenz et al. [6] overlaps with our research as both studies aim to answer exploratory research questions about REST design practices. Moreover, they use thematic analysis to identify relevant concepts. However, our data sources differ. Coblenz et al. [6] use interviews to capture the opinions of API designers on the challenges they face, whereas we rely on online API documentation to observe the end result implementation.

In order to access ML resources, cloud platforms may provide Application Programming Interfaces (APIs) based on HTTP methods. The design of these interfaces is guided by REST design practices. Petrillo et al. [22] collects cloud service offerings and 73 best practices in REST from previous works. The authors then analyze various cloud APIs for understandability and reusability. They conclude that the analyzed APIs achieve an acceptable level of maturity. Our research overlaps with the study by Petrillo et al. [22] in its analysis of REST rule compliance on cloud APIs. However, our focus is specifically on ML cloud APIs, resulting in distinct study objects. Similarly, both studies employ exploratory research questions. Despite the methodological differences, both our research and Petrillo et al. [22] analyze API documentation.

Rodríguez et al. [24] conducted a large-scale analysis of HTTP traffic generated by a mobile internet provider. The study focused on identifying and evaluating best practices related to resource representation and addressability, interface uniformity, statelessness, and hypermedia use. The findings revealed that GET and POST methods dominated the majority of the traffic. Additionally, the most common error response codes were 403 (Forbidden) and 404 (Not Found), suggesting frequent attempts to access restricted or non-existent resources. In terms of API maturity levels, the study found that most APIs achieved level 2, which relies on HTTP verbs. However, only a few APIs reached level 3, which incorporates hypermedia. Although Rodríguez et al. [24] also analyze REST rule compliance, their approach differs. They analyze internet traffic instead of manually checking API endpoints through documentation and HTTP requests. Consequently, our study can investigate rule adherence in more depth, but provides a less comprehensive overview of how APIs implement REST practices across the internet. Additionally, Rodríguez et al. [24] do not explore new potential design practices.

Several tools have been developed to detect design rule violations. One such example is the framework developed by Moha et al. [17] for identifying antipatterns in service-oriented architecture (SOA). These antipatterns are assessed based on factors like service cohesion, coupling, and dependencies. Examples include returning small data in primitive types, which encourages clients to make

multiple requests, and the existence of service chain dependencies, which can hinder flexibility and efficiency. The authors also highlight underutilized or rarely accessed services as another antipattern. In our research, we focus on design practices that can be applied to SOAs to enhance the understandability and reusability of services. Therefore, the antipatterns discussed by Moha et al. [17] might correspond to REST rule violations, but at a higher level of abstraction.

Moving to a more specific level, Palma et al. [21] developed a framework for identifying antipatterns in REST APIs. This framework includes a list of design patterns, along with their corresponding antipatterns and distinctive characteristics. The tool aims to help identify potential pitfalls and improve system usability and maintainability. Similarly, Bogner et al. [3] developed a tool that detects violations of HTTP API design rules. By utilizing natural language processing (NLP) techniques, this tool thoroughly analyzes OpenAPI specifications. However, it's important to note that these tools might not identify all rule violations, such as those related to improper usage of HTTP verbs or hypermedia.

Atlidakis et al. [2] describe a REST rule checker designed to detect rule violations that can lead to API security risks. These risks include access to deleted resources, side effects from failed creations, and unauthorized access. The rule checker relies on algorithms that stress API endpoints by performing CRUD (Create, Retrieve, Update, Delete) operations to identify potential security vulnerabilities. The authors conclude that web APIs deployed in Azure and Microsoft Office 365 services contain several security risks due to the studied rule violations.

Several key studies have proposed automated REST rule violation detectors [17][21][3][2]. These differ from our methodology in two ways. First, they focus on automation, whereas we manually check API rule violations on specific web APIs. Second, they do not explore common design practices within a defined group of web APIs, which is a key focus of our research. However, these studies provide valuable insights into the potential consequences of rule violations, such as security threats [2] or reduced API understandability [3].

Finally, the study by Neumann et al. [19] examines a set of public APIs from popular websites to assess their adherence to REST architectural principles and design practices. They additionally explore common design choices for developing RESTful web services. In this regard, Neumann et al. [19] shared similar goals with our research, as both studies aim to identify commonalities among APIs and analyze their compliance with REST guidelines. However, our work employs a systematic methodology to identify these commonalities within a smaller set of APIs offered by cloud providers. Neumann et al. [19] revealed that 19% of the APIs lack version information, only 4.2% comply with HATEOAS, and 22% of the APIs solely utilize GET and POST HTTP verbs. Overall, while JSON does have widespread support among the analyzed APIs, only 0.8% of services strictly complied with all REST principles.

### 3 STUDY DESIGN

Both research questions were addressed using qualitative methods. RQ1, an exploratory question, was investigated through an inductive thematic analysis approach [7]. This method and the data collection strategy were further explained in section 3.2. RQ2 was



addressed through a manual analysis of REST rule compliance in the cloud APIs, as detailed in section 3.3. Section 3.1 describes each of the MLaaS APIs selected for this study. For transparency and replicability, we share our study artifacts on Figshare<sup>2</sup>.

### 3.1 Objects

Five ML cloud APIs from five distinct ML platforms were selected: Google Cloud, Amazon Web Services (AWS), Azure, IBM Cloud (major players), and BigML (a smaller platform). In Figure 1, ML cloud providers were compared in terms of generated revenue.

For the selection of APIs, it was considered whether the cloud platform offered all necessary services for building and deploying an ML model. This includes services for storing and managing datasets, creating and training ML models, evaluating their performance, and making predictions. Other platforms like H2O or DeepCognition also offer ML services, often tailored to specific functionalities such as document scanning or optical character recognition (OCR). Those APIs that offer a complete set of ML operations were selected.

**3.1.1 Google Cloud.** The Google Cloud platform provides a suite of services built on modular components. It allows users to access IaaS, PaaS, and serverless computing environments. Within this platform, the VertexAI API has been chosen for further study. The VertexAI API offers functionalities for both generative AI and traditional ML operations. Specifically, this study focuses on the ML services used to build ML models.

**3.1.2 AWS.** AWS offers on-demand cloud computing platforms and APIs on a pay-as-you-go basis. Amazon SageMaker, a selected API from this provider, is built for ML projects. It provides a comprehensive set of packaged solutions to prepare, build, train, and deploy ML models.

**3.1.3 Azure.** Azure offers diverse capabilities to meet various business needs. Users can manage, access, and develop applications and services on this platform. Azure provides a wide range of service models, including SaaS, PaaS, and IaaS. The chosen API from Azure was Azure ML, which offers functionalities for building and scaling ML infrastructure.

**3.1.4 IBM Cloud.** IBM Cloud is a suite of cloud computing services tailored specifically for businesses. Offering a wide range of capabilities, it includes compute, storage, networking, database management, analytics, ML, and tools for developers. Designed for handling large datasets, the chosen API from IBM Cloud was IBM ML, which empowers users to train and deploy ML models.

**3.1.5 BigML.** BigML is an ML platform that offers programmability and scalability. It simplifies the process of solving and automating ML models and operations. The API selected for study is the only product offered by the provider, the BigML API.

### 3.2 Existing practices for designing and specifying web APIs for ML cloud services

An inductive approach was followed to reduce the risk of obtaining preconceived results. Data was constantly reviewed and compared, selecting text segments from online documentations and assigning

codes that define emerging concepts. The whole process of thematic analysis is expressed by Figure 2.

**3.2.1 Data collection.** Web pages containing documentation about the ML cloud APIs offered by various providers were downloaded, along with the date and time of access. Only official documentation provided at the platform's websites was collected. This initial online research served to narrow the focus towards the main topic of interest: REST design practices within these APIs. The downloaded documentation provided a broad overview of the different areas of ML in cloud services, including their interactions with users and the cloud platform itself. Subsequently, a more in-depth examination of the API documentation was conducted to identify recurring design practices employed across the different cloud APIs.

To ensure objectivity in addressing RQ1, which centers on the design and functionality of the APIs, two researchers independently extracted relevant data. Initially, the official documentation for each selected API was downloaded in PDF format. When documentation spanned multiple web pages, the most relevant ones were downloaded. These documents were then sent to another researcher for simultaneous annotation. After reviewing all documentation, the researcher returned annotated PDFs. Text snippets were compared to identify discrepancies, which were collaboratively resolved through online discussions. Finally, PDF annotations were transcribed into an Excel sheet containing annotation texts and corresponding codes.

**3.2.2 Coding.** In qualitative studies, coding refers to the process of uncovering themes from data to build theory. This systematic approach involves categorizing the data. It is typically organized into three phases [26]: open coding, axial coding, and selective coding. During open coding, initial themes are identified from the data. In axial coding, these themes are refined and relationships between them are established. Finally, selective coding integrates the categories into a cohesive theory with meaning.

For this study open coding was initially used for getting an overview of the main components of MLaaS platform. As the open coding progressed, the scope of interest narrowed towards REST APIs in MLaaS platforms. Knowing then what were the functionalities of REST interfaces and which other elements of a cloud platform they interact with. Then, in axial coding, the codes were organised into categories obtaining the different concepts to look at when comparing the APIs, and from where they emerge from. All these concepts came together resulting into a theory which explains how ML APIs are designed and which commonalities and differences they have.

**3.2.3 Data analysis.** Thematic analysis was the chosen technique for building a theory to address RQ1. It encompasses the entire process, from data extraction to the generation of a conceptual model that represents the newly developed theory. Coding is a crucial step within thematic analysis. As described by Naem et al. [18], thematic analysis involves several steps: selecting statements from the data, extracting keywords from those statements, coding, theming, conceptualization, and finally, the creation of a conceptual model. Theming refers to grouping coded elements into meaningful categories. In this study, as explained in section 3.2.2, theming was integrated within the coding phase.

<sup>2</sup><https://figshare.com/articles/software/master-thesis/26521441?file=48287056>

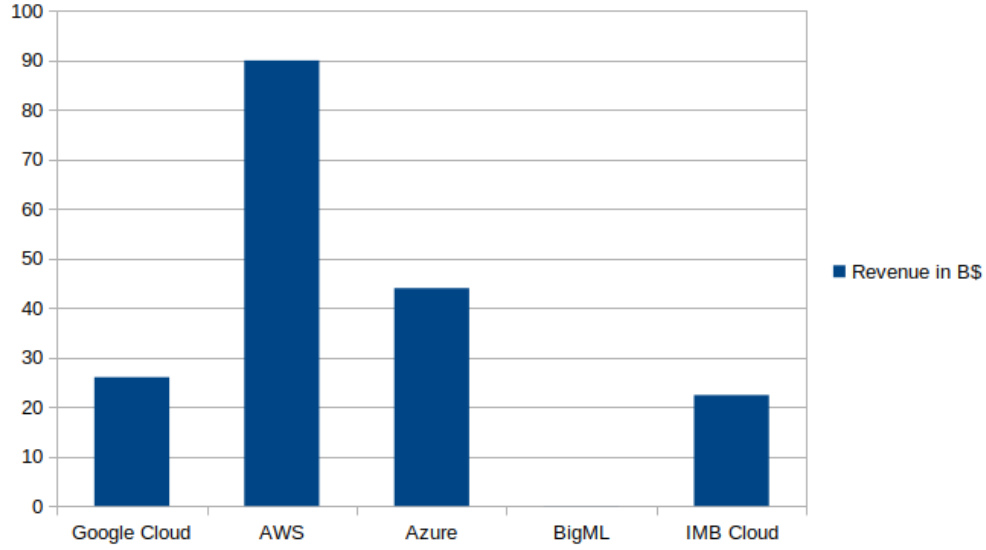


Figure 1: ML Cloud Platform revenues in 2023

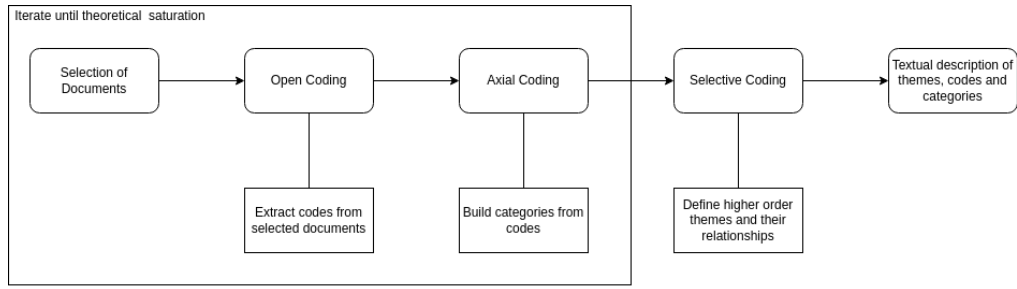


Figure 2: Thematic Analysis research process

Conceptualization involves interpreting and formalizing the underlying ideas and concepts that emerge from the identified themes. However, Cruzes et al. [7] proposed an alternative approach better suited for software engineering studies. This approach emphasizes the creation of a conceptual model using higher-order themes. These higher-order themes represent broader categories encompassing lower-level concepts. Then, we can then further examine the relationships between these higher-order themes, their connection to existing theory, and finally synthesize this information to create a comprehensive conceptual model.

For this research, higher-order themes were created to identify the main elements of cloud platforms offering ML services. These themes helped uncover both functional and quality requirements considered by ML users. Subsequently, the research analyzed the impact of these requirements on API design. Additionally, it investigated which cloud infrastructure elements REST APIs interact with, and for which purposes are they used.

### 3.3 ML cloud APIs adherence to REST API design rules

For each API, a selection of endpoints was used to conduct the manual analysis. Each endpoint was assessed against a set of REST rules for compliance. This process is illustrated in Figure 3.

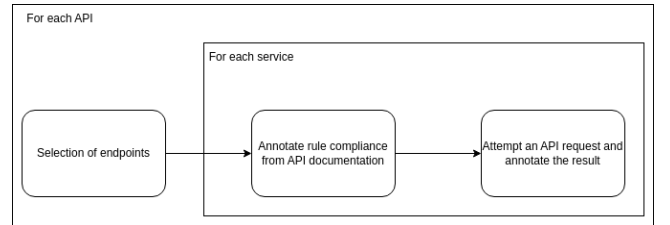


Figure 3: API REST compliance research process

**3.3.1 Data collection.** The data used for answering RQ2 comes from two different sources. One is the online documentation provided by ML platforms containing the API specifications. The other

source is the endpoints from the APIs, which can be accessed by subscribing to the MLaaS platforms.

**3.3.2 Selection of endpoints.** For each MLaaS API, a set of endpoints was selected, those offering functionalities to perform ML operations. These operations typically involve tasks like creating models, storing datasets, or setting data sources. Each endpoint was then used to assess whether its interface adheres to the REST design rules proposed by Massé [16]. Additionally, web pages containing documentation about the analysed endpoints were also consulted. In this study, key endpoints essential for an ML workflow were included in the analysis. These endpoints are typically used for managing dataset storage, creating and training an ML model, and managing pipelines. These endpoints enable the creation of an ML workflow hosted in the cloud infrastructure and are offered by all researched providers.

**3.3.3 Data analysis.** To determine a cloud platform’s compliance with REST design rules, both online API documentation and API endpoints were analyzed. To identify evidence within the documentation, specific sections were annotated to highlight potential rule violations. Additionally, a set of HTTP requests executed using Postman<sup>3</sup> provides empirical evidence on rule compliance. These requests expose details like HTTP verbs, response status codes, request/response headers, and request parameters.

A rule was considered violated if an analysis of the API documentation or attempted API requests reveals an endpoint that clearly fails to adhere to the rule. Conversely, if all examined examples comply with the rule and none violate it, the API was considered compliant. In cases where no examples of either compliance or violation were found, the rule compliance remains undetermined. There may be cases in which the rule was partially complied, as some endpoints of an API adhere to the rule while some others fail to adhere.

**3.3.4 Selection of REST rules for analysis.** A subset of the REST rules proposed by Massé [16] were selected for analysis. Among these selected rules, 21 are categorized as high importance, and 9 are categorized as medium importance according to Kotstein et al. [15]. From all the rules compiled by Kotstein et al. [15], some rules were excluded due to the difficulty of validating them without access to the API source code or their irrelevance to cloud ML APIs.

## 4 RESULTS

The study execution is divided into two subsections. At section 4.1 the results from thematic analysis are presented. Starting with the higher-order conceptualization of themes, section 4.1 delves into REST APIs within MLaaS platforms, discussing their implementation through HTTP methods, headers, and URIs, while contrasting API characteristics concerning subdomains, paths, query parameters, and resource modification. Furthermore, it compares API designs by evaluating hypermedia implementation, API versioning strategies, and error-handling practices. At section 4.2 the results from the manual analysis on REST rule compliance per API are presented.

<sup>3</sup><https://www.postman.com/>

### 4.1 Existing practices for designing and specifying web APIs for ML cloud services

Thematic analysis identified seven themes, from which two are considered subthemes. A subtheme is simply a more specific concept that falls under a broader theme. Each theme, or subtheme, represents a higher-order concept derived from a set of codes categorized during the coding phase. The themes are interrelated, indicating how elements within one theme interact with elements from another. For example, resource management elements like ML pipelines interact with ML resources such as ML models. In this case, ML pipelines orchestrate and manage ML models. A visual representation of the relationships between themes can be found in Appendix A, and important code definitions in Appendix B. The subtheme REST API, which is the main focus of analysis in this study, is presented in section 4.1.1. Each other theme is described as follows:

- **Infrastructure** represents the computing resources and services offered by MLaaS platforms. These include hardware, software, and network resources. MLaaS platforms leverage distributed computing for scalability, allowing them to handle changing workloads. A quality requirement relevant for cloud infrastructures is interoperability, the ability of different systems and platforms to work together seamlessly (ISO/IEC 25000, 2014).
- **Security** refers to the practices and protocols users can employ to secure their cloud infrastructure. MLaaS platforms often rely on shared responsibility models, where both users and providers play a role. Users can secure their environment by setting access policies and employing authentication protocols like OAuth2. Data backup functionalities are also available to recover lost information.
- **ML Resources** are the core components of machine learning projects. These include supervised, unsupervised, and ensemble models, each serving specific purposes. Training data is typically accessed through URLs pointing to cloud storage, avoiding large files overloading API requests. Inferences, predictions made by trained models, can be accessed in real-time or through batch processing. Real-time inference is suitable for low-latency requests because it provides synchronous access to predictions. On the other hand batch inference is ideal for high-latency, computationally demanding predictions. It allows creating asynchronous prediction jobs.
- **Resource Management** analyzes the management of ML resources. Examples of ML resources include models and datasets. Normally, each resource has a unique identifier, which is provided within an API request. APIs are a common interface for managing these resources. Additionally, functionalities for automating the entire ML workflow (MLOps) are often available. This includes building pipelines for model creation and deployment, as well as monitoring and version control.
- **Developer Resources** facilitate application development. Platforms offer pre-built development environments with popular libraries and frameworks, often focusing on Python. Containerization of services is becoming a prevalent trend.



Additionally, AI studios enable code-free ML operations and pre-built resources like datasets and models can jumpstart development efforts.

- Interface examines how users interact with MLaaS platforms. Common interfaces include CLIs for administrative tasks, SDKs for application development, and REST APIs for programmatic interaction. While CLIs and SDKs offer user-friendly experiences, REST APIs provide the foundation for automation and integration offering a standardized HTTP-based approach.

**4.1.1 REST API.** As a subtheme of Interface, this theme explores REST APIs, one of the mechanisms through which users can interact with the MLaaS platform. REST APIs act as client-side communication gateways, enabling users to access ML resources or perform resource management tasks. For each ML resource, a different endpoint is exposed by the platforms. The types of endpoints used by MLaaS platforms in their API offerings can be observed in Table 1.

This client-server communication leverages the HTTP protocol. Our analysis focuses on various aspects of HTTP exchanges, including both HTTP requests and responses. For HTTP requests, three subcategories emerged (URI subdomain, URI path, and query parameters), all concerning how information is passed to the API server.

Furthermore, HTTP headers can also be used to transmit and receive metadata. On the other hand, HTTP verbs implicitly convey the action the server should perform within the request. The design of these elements in an HTTP exchange significantly impacts the API's compatibility with client applications or services, ultimately enhancing or hindering its reusability and understandability.

A visual representation of the theme categories, expressed as a hierarchical relationship, can be found in Figure 4. Table 2 contains annotations, derived codes, and their corresponding provider.

API compatibility encompasses two main subcategories: API migrations and API versioning. API migrations involve developers transitioning from one ML API to another. API versioning helps developers avoid incompatibility issues arising from new API versions that introduce significant changes. Due to the rapid evolution of the field, this category holds immense relevance for both providers and MLaaS users, as ML APIs constantly undergo changes and updates [20].

Moreover, in HTTP responses, a good practice is to leverage hypermedia (provided as URI paths) to link resources and guide clients towards their next actions. Two primary implementations of hypermedia can be found in ML REST APIs:

- Linking Data Entities: This approach links data entities, such as datasets, to related ML resources like models.
- Hypermedia as the Engine of Application State (HATEOAS): This practice involves providing information about the state of certain ML resources.

Additionally, a set of characteristics derived from the categories are grouped together, and compared per provider. They have been formatted into "Yes" or "No" questions and have a category assigned. The characteristics are not either design patterns or antipatterns, they are just the results from observations. The table displaying the API characteristics and their adoption from the MLaaS platform is shown in Table 3.

The API characteristics obtained from thematic analysis shows that Google Cloud uses "api" in the subdomain, while neither Azure, IBM nor BigML uses "api" in the subdomain. None of the cloud providers use "api" in the path. Both Google Cloud and Azure use an identifier for a resource container or group in the path, IBM does it in the query parameters, and BigML does not require it. Google Cloud and IBM specify the server location in the subdomain, while neither Azure nor BigML specify the server location in the subdomain. Only Google Cloud specifies the server location in the path. Moreover, Google Cloud and Azure use two or more identifiers in the path to specify a resource, while BigML and IBM do not use two or more identifiers. IBM uses snake\_case for query parameters, and Google Cloud uses camelCase. BigML and Google Cloud use access tokens or API keys in request parameters, while Azure and IBM include access tokens or API keys in the request headers.

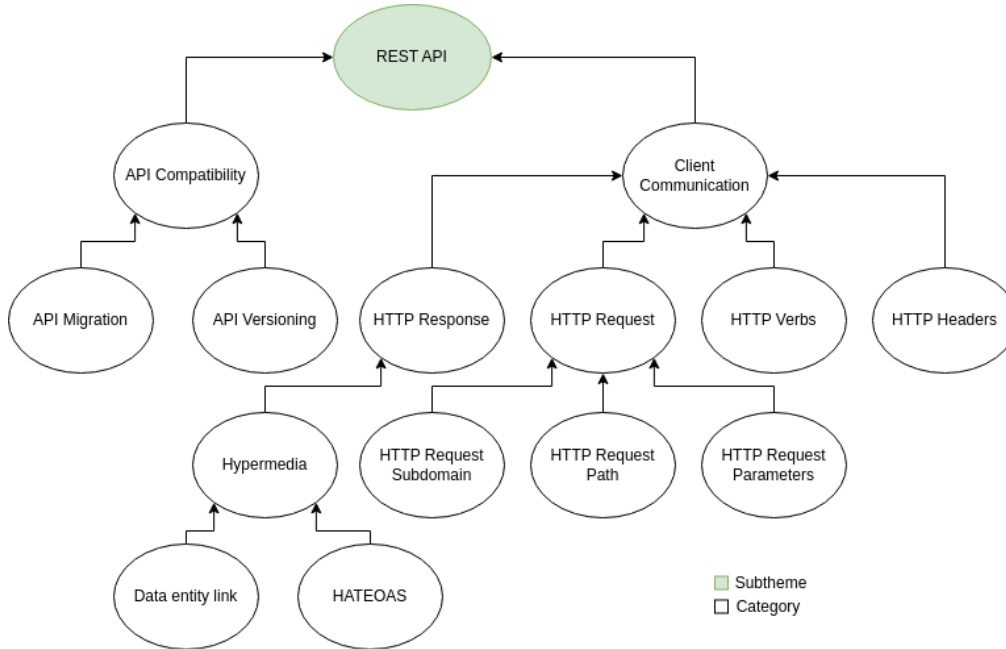
It is also noticeable that the requirement and format for specifying the API version differs across major cloud providers. Google Cloud, Azure, and IBM all mandate that users specify a version of the API they are using. Google Cloud and IBM use a format of "v{version\_number}" to denote the API version, such as "v1" or "v2". Azure, on the other hand, utilizes a date format to indicate the API version. In contrast, the BigML cloud platform does not require users to specify a version of the API at all. Instead, BigML employs a string-based format for versioning, without necessitating explicit version declaration from users interacting with their services. Only Azure requires specifying the API version in the query parameters, Google Cloud BigML and IBM require it in the path. A comparison between two HTTP request used for retrieving a representation of a dataset from the BigML and the G. Vertex AI APIs can be observed at Figure 5.

Differences have also been observed in the handling of forbidden requests. IBM ML returns a 404 Not Found status code, which is technically incorrect as the resource exists but access is forbidden, while Azure ML uses a 401 Unauthorized status code, indicating that the user is not authenticated or lacks necessary permissions. G. Vertex AI employs the appropriate 403 Forbidden status code, explicitly stating that the user is forbidden from accessing the resource. In terms of response body structure, IBM ML provides minimal information while Azure ML offers a more informative response with error code and message. G. Vertex AI delivers the most detailed response, including error code, message, permission details, trace ID, and metadata. A comparison of a forbidden request error handling is presented at Figure 6. The forbidden requests where made from a platform's user account different than the account who has the access rights to the ML resource.

When it comes to API resource updates, IBM uses a JsonPatch-Operation format in request body payloads for updating resources. JsonPatchOperation payloads are structured by specifying the operation to be performed, the value to be updated, and the path that identifies the resource. In contrast with providing the resource representation as it is, this approach allows to send only the required data and improves on efficiency and consistency among API update operations. On the other hand, there exists a trade off in terms of understandability, as JsonPatchOperation adds complexity to request payloads. Only Azure uses the PUT verb for both creating and updating resources. An example of a JsonPatchOperation HTTP request body can be observed at Figure 7.

**Table 1: MLaaS API endpoint offering comparison**

Endpoint Type	Functionality	G. Vertex AI	Azure ML	BigML	IBM ML	A. Sagemaker
Dataset	Access a Dataset resource	Yes	Yes	Yes	Yes	Yes
Data Source	Access the location of a Dataset resource	No	Yes	Yes	No	No
Dataset Version	Manage different Dataset versions	Yes	Yes	No	No	Yes
Model	Access an ML model resource	Yes	Yes	Yes	Yes	Yes
Model Version	Manage different ML model versions	Yes	Yes	No	No	Yes
Jobs	Tasks running on the cloud generally for training ML Models	Yes	Yes	No	Yes	Yes
Tuning	Find the best parameters for an ML model	Yes	No	No	No	Yes
Evaluation	Evaluate performance of an ML model	Yes	No	Yes	No	No
Pipeline	Manage ML operations in a workflow	Yes	Yes	No	Yes	Yes
Metadata	Access ML model metadata	Yes	Yes	No	No	Yes
Deployment	Manage the deployment of an ML model	Yes	Yes	Yes	Yes	Yes
Batch Inference	Perform batch predictions	Yes	Yes	Yes	No	Yes
Workspace	Access groups of ML resources	No	Yes	Yes	No	Yes



**Figure 4: REST API hierarchy of categories**

Retrieve Dataset Endpoint				
API	BigML			G. Vertex AI
URI	https://bigml.io/andromeda/dataset/665c9c601c3c406a907458ac?username=RRUIZSALVAT			https://us-central1-aiplatform.googleapis.com/v1/projects/143597657071/locations/uscentral1/datasets/3200136289223442432/operations/7072043705930088448
HTTP Verb	GET			GET
Query Params	Username, API key			Access token
Path Params	API version, dataset ID			API Version. Project, dataset, and operation IDs. Location name of the cloud server.
API Version Format	String-based			v{version_number}
API Version Required	No			Yes
Response Status Code	200			200
Response Headers	Server	BigML		
	Date	Sun, 02 Jun 2024 16:25:03 GMT		
	Content-Type	application/json		Content-Type application/json; charset=UTF-8
	ETag	202406021625031717345503		Vary Origin
	Cache-Control	private		Vary X-Origin
	Cache-Control	max-age=0, no-cache, no-store, must-revalidate		Vary Referer
	Access-Control-Allow-Origin	*		Content-Encoding gzip
	Access-Control-Allow-Methods	POST, GET, PUT, DELETE		Date Sat, 01 Jun 2024 11:18:40 GMT
	Vary	Accept-Encoding		Server ESF
	Content-Encoding	gzip		Cache-Control private
	X-UA-Compatible	IE=Edge		X-XSS-Protection 0
	X-XSS-Protection	1; mode=block		X-Frame-Options SAMEORIGIN
	X-Content-Type-Options	nosniff		X-Content-Type-Options nosniff
	X-Frame-Options	SAMEORIGIN		Alt-Svc h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
	Via	1.1 google		Transfer-Encoding chunked
	Alt-Svc	h3=":443"; ma=2592000,h3-29=":443"; ma=2592000		
	Transfer-Encoding	chunked		
Response Body	<pre>{   "all_fields": true,   "category": 0,   "cluster": null,   "cluster_status": false,   "code": 200,   ... }, "fields": {   "000000": {     "column_number": 0,     "datatype": "double",     "name": "sepal length",     "optype": "numeric",     "order": 0,     "preferred": true,     "summary": {       "bins": [         [           4.3,           1         ],         [           4.425,           4         ],         ...       ]     },     "number_of_anomalies": 0,     "number_of_anomalyscores": 0,     "number_of_associations": 0,     ...   } }</pre>			<pre>{   "name": "projects/143597657071/locations/ ...",   "metadata": {     "@type": "type.googleapis.com/ ...",     "genericMetadata": {       "createTime": "2024-06-01T11:05:05.279364Z",       "updateTime": "2024-06-01T11:05:16.217984Z"     }   },   "done": true,   "response": {     "@type": "type.googleapis.com/ ...",     "name": "projects/143597657071/locations/ ...",     "displayName": "test_dataset",     "metadataSchemaUri": "gs://google-cloud-aiplatform/ ...",     "labels": {       "aiplatform.googleapis.com/dataset_metadata_schema": "IMAGE"     },     "metadata": {       "dataItemSchemaUri": "gs://google-cloud-aiplatform/ ..."     },     "description": "test dataset"   } }</pre>

Figure 5: Comparison of two HTTP requests for retrieving a Dataset representation

**Table 2: Annotations and codes for the REST API subtheme**

Annotation	Code	MLaaS API
Use real-time inference for a persistent and fully managed endpoint (REST API) that can handle sustained traffic.	REST API	A. Sagemaker
This API uses standard HTTP response codes to indicate whether a method completed successfully.	HTTP response codes	IBM ML
The REST API uses HTTP verbs in a standard way to create, retrieve, update, and delete resources.	HTTP verbs	Azure ML
Each service provider updates their API on their own cadence, but does so without breaking existing programs.	API versioning	Azure ML
Changes to existing API versions are designed to be compatible with existing client applications.	API versioning	IBM ML
The BigML.io API is versioned using code names instead of version numbers.	API versioning	BigML
If you omit the version name in your API requests, you will always get access to the latest API version.	API versioning	BigML
To create or overwrite a named compute resource, you’ll use a PUT request.	PUT request	Azure ML
patch PATCH /v1/dataset.name	PATCH request	G. Vertex AI
Using a data_asset, requires an href to be supplied to the location object whereas using a connection_asset requires the connection_id for the connection object and different location fields depending on the data source type.	Hypermedia data entity link	IBM ML
metadataSchemaUri: Immutable. Points to a YAML file stored on Google Cloud Storage describing additional information about the Model, that is specific to it.	Hypermedia as state engine	G. Vertex AI
ModelDataUrl The S3 path where the model artifacts, which result from model training, are stored.	Hypermedia as state engine	A. Sagemaker

In addition, Figure 7 shows a comparison between two HTTP request from BigML and IBM ML for updating a dataset resource. BigML uses the HTTP method PUT with a request body containing a JSON object. The response code is 202 (Accepted), and the response body includes detailed information about the updated resource, such as “category”, “code”, “created”, “credits”, “dataset”, etc. On the other hand, IBM ML uses the HTTP method PATCH with a request body containing a JsonPatchOperation object. The response code is 200 (OK), and the response body provides information about the updated resource, including “metadata” and “entity”.

Moreover, Google Cloud, BigML, and IBM include URIs in HTTP responses for linking to data sources, while Azure does not. Google Cloud and AWS include URIs in HTTP responses for linking to the status page, but Azure, BigML, and IBM do not. A comparison for creating an ML model resource between the Azure ML and IBM ML APIs is presented in Figure 8. As visible, both Azure ML and IBM ML return the resource’s metadata as a nested JSON object in the HTTP response body. Conversely, the Google Vertex AI API includes a link to metadata in the JSON field “metadataSchemaUri,” as exhibited in Figure 5.

Finally, the handling of bad requests with malformed JSON bodies was investigated. G. Vertex AI, BigML, and IBM ML demonstrate

varying approaches to handling malformed requests. While G. Vertex AI returns a generic 500 Internal Server Error, BigML and IBM ML offer more specific 400 Bad Request responses. IBM ML further enhances error clarity by providing additional trace information. This diversity in error handling highlights the importance of examining how different APIs communicate errors to users for effective debugging and troubleshooting. Figure 9 shows the comparison between these three APIs when responding to a malformed request.

## 4.2 ML cloud APIs adherence to REST API design rules

The outcomes of the manual analysis are detailed in Table 4. For each API, 30 REST rules have been validated with both the documentation and HTTP requests. The table summarizes the outcomes of these validations. It uses the following markings:

- “Adhere”: If the rule is clearly fulfilled in both the documentation and the set of HTTP requests.
- “Violate”: If the rule is clearly violated for at least one endpoint from the API and it is visible either in the documentation or in the set of HTTP requests.

**Table 3: API characteristics derived from the REST API theme**

Category	Characteristic	G. Vertex AI	Azure ML	BigML	IBM ML	A. Sagemaker
API Versioning	Is specifying version is required?	Yes	Yes	No	Yes	-
API Versioning	Is API Version in "v{version_number}" format?	Yes	No	No	Yes	-
API Versioning	Is API Version in date format?	No	Yes	No	Yes	-
API Versioning	Is API Version in string format?	No	No	Yes	No	-
HTTP Request Path	Is API Version in the path?	Yes	No	Yes	Yes	-
HTTP Request Parameters	Is API Version in the query parameters?	No	Yes	No	Yes	-
HTTP Request Subdomain	Is "api" in the subdomain?	Yes	No	No	No	-
HTTP Request Path	Is "api" in the path?	No	No	No	No	-
HTTP Request Path	Is there an identifier of a resource container or group in the path?	Yes	Yes	No	No	-
HTTP Request Parameters	Is there an identifier of a resource container or group in the query parameters?	No	No	No	Yes	-
HTTP Request Subdomain	Is the server location in the subdomain?	Yes	No	No	Yes	-
HTTP Request Path	Is the server location in the path?	Yes	No	No	No	-
HTTP Request Path	Does it use 2 or more identifiers in the path?	Yes	Yes	No	No	-
HTTP Request Parameters	Does it use snake_case in the query parameters?	No	No	No	Yes	-
HTTP Request Parameters	Does it use camelCase in the query parameters?	Yes	No	No	No	-
HTTP Request Parameters	Are access tokens or API keys included in request parameters?	Yes	No	Yes	No	-
HTTP Headers	Are access tokens or API keys included in request headers?	No	Yes	No	Yes	-
HTTP Verbs	Does it use the PATCH verb for partial updates?	Yes	No	No	Yes	-
HTTP Verbs	Does it use the PUT verb for both creating and updating resources?	No	Yes	No	No	-
HTTP Verbs	Does it use JsonPatchOperation for updating resources?	No	No	No	Yes	-
Hypermedia data entity link	Does it include URIs in HTTP responses for linking to data sources?	Yes	No	Yes	Yes	Yes
Hypermedia as state engine	Does it include URIs in HTTP responses for linking to the state?	Yes	No	No	No	Yes



Model List Forbidden Responses			
API	IBM ML	Azure ML	G. Vertex AI
Response Status Code	404	401	403
Response Body	<pre>{   "trace": "bhowp29mqy74",   "errors": [     {       "code": "project_not_found",       "message": "Failed to find project 98a35d75-c ..."     }   ] }</pre>	<pre>{   "error": {     "code": "InvalidAuthenticationTokenTenant",     "message": "The access token is from the wrong issuer ..."   } }</pre>	<pre>{   "error": {     "code": 403,     "message": "Permission 'aiplatform.models.list' denied on resource '//aiplatform. ...'",     "status": "PERMISSION_DENIED",     "details": [       {         "@type": "type.googleapis.com/google.protobuf.StringValue",         "value": "IAM_PERMISSION_DENIED",         "reason": "aiplatform.googleapis.com",         "domain": "aiplatform.googleapis.com",         "metadata": {           "aiplatform.models.list": {             "permission": "aiplatform.models.list",             "resource": "projects/ ..."           }         }       }     ]   } }</pre>

**Figure 6: Comparison of three HTTP requests to a forbidden resource**

- “Partially Adhere”: When the rule is generally fulfilled but some exceptions are found for some of the analysed endpoints.
- “Undetermined”: When it cannot be determined whether the rule is fulfilled or not.

Google Cloud, Azure, BigML, and IBM seem to adhere to most of the rules, with a few exceptions. Among them, Azure is the most compliant, while IBM exhibits the most partial adherence. For AWS, most rules remain undetermined because the SageMaker API is offered in packaged SDKs. These SDKs encapsulate HTTP requests within client libraries for use in application code. Consequently, for AWS, the only data source is documentation about SDK methods, which implicitly indicates API design, such as the use of response status codes.

Rule 1 is followed by most platforms except for BigML, for which it remains undetermined because it does not have hierarchical relationships. Rule 2, avoiding a trailing forward slash, is followed by all APIs. Moreover, all APIs use variable path segments to identify resources (rule 13). As observed in section 4.1.1, some APIs use multiple identifiers in the path. This is often because not only a resource identifier must be specified, but also the server location or the resource group to which it belongs. All APIs use query parameters (rule 15), specifically for list requests. However, some APIs use query parameters to specify the API version instead.

Rule 46, content type must be set, and rule 61, JSON must be used, are implemented by all APIs and seem to be a well-established standard. On the HTTP verbs, GET to retrieve the representation of a resource, and DELETE to delete a resource, seem to unanimously

adhere to REST rules 18 and 24. For HTTP status codes, the 200 for non-specific success and 401 for responding to unauthorized access are also unanimously well applied as indicated by rules 26, 27 and 37. Rule 73, consistent forms in error responses is also followed by all APIs except for AWS, that remains undetermined.

Both Google Cloud and AWS follow REST rules 38 and 45, providing 403 status codes for forbidden requests and 500 for internal server errors. Azure does not follow the rule of using 403 for forbidden access, but adheres to the rule for indicating internal server errors. BigML and IBM both partially adhere to both REST rules, though for most of the APIs it could not be determined the adherence of rule 45.

On the other hand, some design rules bring more discrepancies between the APIs. On URI design, rule 3, the use of hyphens, is not always preferred as some APIs use camelCase. That also impacts on rule 5, the use of lowercase, which is consequently not followed by some APIs.

Although all APIs unanimously use verbs in the URI for controllers, there are discrepancies on the HTTP verb that should be used. Despite rule 23, POST should be used for controllers, some APIs use GET. Furthermore, there are also discrepancies on the use of PUT. As for partial updates some APIs implement the PATCH verb, and for the creation of resources some use PUT when POST should be used instead.

Furthermore, rule 30, using the status code 204 when the response body is left intentionally empty, is violated by some APIs while followed by others when implementing responding to a successful deletion of a resource. On the implementation of metadata

**Table 4: API REST rule compliance**

Category	Importance	Text	G. Vertex AI	A. Sagemaker	Azure ML	BigML	IBM ML
URIs	High	Forward slash separator (/) must be used to indicate a hierarchical relationship	Adhere	Adhere	Adhere	Undetermined	Adhere
URIs	Medium	A trailing forward slash (/) should not be included in URIs	Adhere	Adhere	Adhere	Adhere	Adhere
URIs	High	Hyphens (-) should be used to improve the readability of URIs	Violate	Adhere	Partially Adhere	Violate	Violate
URIs	High	Underscores (_) should not be used in URIs	Adhere	Adhere	Adhere	Adhere	Violate
URIs	High	Lowercase letters should be preferred in URI paths	Violate	Adhere	Violate	Adhere	Adhere
URIs	Medium	File extensions should not be included in URIs	Adhere	Undetermined	Undetermined	Partially Adhere	Adhere
URIs	High	A singular noun should be used for document names	Undetermined	Undetermined	Undetermined	Adhere	Undetermined
URIs	High	A plural noun should be used for collection names	Adhere	Undetermined	Adhere	Violate	Adhere
URIs	Medium	A verb or verb phrase should be used for controller names	Adhere	Undetermined	Adhere	Undetermined	Adhere
URIs	High	Variable path segments may be substituted with identity-based values	Adhere	Adhere	Adhere	Adhere	Adhere
URIs	High	CRUD function names should not be used in URIs	Violate	Undetermined	Adhere	Adhere	Adhere
URIs	Medium	The query component of a URI may be used to filter collections or stores	Adhere	Adhere	Adhere	Adhere	Adhere
HTTP	High	GET must be used to retrieve a representation of a resource	Adhere	Undetermined	Adhere	Adhere	Adhere
HTTP	High	POST must be used to create a new resource in a collection	Adhere	Undetermined	Violate	Adhere	Adhere
HTTP	High	POST must be used to execute controllers	Adhere	Undetermined	Adhere	Violate	Violate
HTTP	Medium	PUT must be used to both insert and update a stored resource	Violate	Undetermined	Adhere	Adhere	Partially Adhere
HTTP	High	DELETE must be used to remove a resource from its parent	Adhere	Undetermined	Adhere	Adhere	Adhere
HTTP	Medium	200 ("OK") should be used to indicate nonspecific success	Adhere	Undetermined	Adhere	Adhere	Adhere
HTTP	High	200 ("OK") must not be used to communicate errors in the response body	Adhere	Undetermined	Adhere	Adhere	Adhere
HTTP	High	201 ("Created") must be used to indicate successful resource creation	Violate	Undetermined	Adhere	Adhere	Adhere
HTTP	High	204 ("No Content") should be used when the response body is intentionally empty	Violate	Undetermined	Violate	Adhere	Adhere
HTTP	High	401 ("Unauthorized") must be used when there is a problem with the client's credentials	Adhere	Violate	Adhere	Adhere	Adhere
HTTP	High	403 ("Forbidden") should be used to forbid access regardless of authorization state	Adhere	Adhere	Violate	Partially Adhere	Partially Adhere
HTTP	High	500 ("Internal Server Error") should be used to indicate API malfunction	Violate	Adhere	Undetermined	Undetermined	Undetermined
Meta	High	Content-Type must be used	Adhere	Adhere	Adhere	Adhere	Partially Adhere
Meta	Medium	Content-Length should be used	Violate	Undetermined	Adhere	Violate	Partially Adhere
Meta	Medium	Location must be used to specify the URI of a newly created resource	Violate	Undetermined	Adhere	Adhere	Violate
Meta	Medium	Caching should be encouraged	Adhere	Undetermined	Violate	Violate	Violate
Repr.	High	JSON should be supported for resource representation	Adhere	Adhere	Adhere	Adhere	Adhere
Repr.	High	A consistent form should be used to represent error responses	Adhere	Undetermined	Adhere	Adhere	Adhere

Dataset Update Endpoint		
API	BigML	IBM ML
HTTP verb	PUT	PATCH
HTTP Request Body	<pre>{   "name": "a new name" }</pre>	<pre>[   {     "op": "replace",     "path": "name",     "value": "new_name"   } ]</pre>
Response Status Code	202	200
Response Body	<pre>{   "all_fields": true,   "category": 0,   "cluster": null,   "created": "2024-06-02T16:22:56.305000",   "creator": "rruizsalvat",   ...    "fields": {     "000000": {       "column_number": 0,       "datatype": "double",       "name": "sepal length",       "optype": "numeric",       "order": 0,       "preferred": true,       "summary": {         "bins": [           [             4.3,             1           ],           [             4.425,             4           ],           ...         ]       }     }   } }</pre>	<pre>{   "entity": {},   "metadata": {     "created_at": "2024-06-07T17:40:37.084Z",     "description": "This is my first resource.",     "id": "e83d9ae3-1c59-43d5-8b89-b38d34a8f2e9",     "modified_at": "2024-06-07T17:40:37.084Z",     "name": "my-resource",     "owner": "IBMId-692000F6R5",     "project_id": "cce6c1cd-30e5-4b16-a380-966592b1a296",     "tags": [       "t1",       "t2"     ]   },   "system": {     "warnings": []   } }</pre>

**Figure 7: *JsonPatchOperation* request body example**

REST rules, both specifying content length and location are not provided in the responses for some of the APIs.

In addition to the results described above, Figure 10 presents the number of REST rules adhered to, partially adhered to, and violated, for each API analyzed. Note that undetermined labelled rules are not considered in the bar chart. The results showed that, on average, 74.19% of the rules are fully compliant, while only 20.16% are violated. Nevertheless, it's important to note that the chart itself may not be a sole indicator of how well the APIs are designed. Violations of certain rules can have a greater impact on usability compared to others.

In addition to the presented results, Figure 11 shows an example of a violation of rule 28 (the status code of a successful resource creation response) by Google Cloud. Figure 12 shows an example of a violation of rule 30 (the status code of an empty response) by Azure, and Figure 13 shows an example of a violation of rule 4 (the unrecommended use of underscores in URIs) by IBM.

## 5 DISCUSSION

As cloud computing gains traction and cloud based web services are becoming a standard in the industry, cloud based architectures influence the design of web APIs [28]. Nevertheless, REST rules were proposed by Massé [16] in 2011, when cloud computing was

not as popular as it is nowadays [12]. After conducting the thematic analysis study, we believe that REST rules should get adapted to new trends and technologies used for the design of web APIs. For instance, this could involve considering the use of PATCH HTTP verbs for partial updates and JsonPatchOperation formats in request body payloads.

From Table 6, a difference is noticeable in how IBM, Azure, and Google Cloud handle requests with forbidden access to a resource. For future research, it would be helpful to determine which design choice is best in terms of security. On the one hand, returning a 403 Forbidden status code enhances API understandability and facilitates debugging. However, it also reveals the existence of a resource. On the other hand, returning a 404 Not Found status code might hinder debugging while obscuring the resource's existence from potential attackers.

Thematic analysis revealed advantages for utilizing hypermedia to connect ML resources and facilitate developers in programmatically automating ML workflows. As a consequence, most APIs try achieving level 3 of maturity from Richardson's maturity model [10]. However, some of the rule violations in which APIs occur are related with the use of HTTP verbs (level 2 from Richardson's maturity model [10]). For this reason we believe that a review on the API maturity model would be convenient for assessing the maturity

Create Model Endpoint																																																																														
API	Azure ML	IBM ML																																																																												
URI	https://management.azure.com/subscriptions/414c2448-40a3-428c-8a64-70a49b8415a7/resourceGroups/rogerresourcegroupthesis/providers/Microsoft.MachineLearningServices/workspaces/rogerworkspacethesis/models/testContainer?api-version=2024-04-01	https://eu-gb.ml.cloud.ibm.com/ml/v4/models?version=2020-09-01																																																																												
HTTP Verb	PUT	POST																																																																												
Query Params	API Version (date format)	API Version (date format)																																																																												
Path Params	Subscription ID, resource group name, provider service, workspace and resource (model) name	API Version (v{version_number} format)																																																																												
Request Headers	Access token	Access token																																																																												
Request Body	<pre>{  "properties": {    "description": "Model container description",    "tags": {      "tag1": "value1",      "tag2": "value2"    }  }</pre>	<pre>{  "name": "my-resource",  "project_id": "cce6c1cd-30e5-4b16-a380-966592b1a296",  "description": "This is my first resource.",  "tags": [    "t1",    "t2"  ],  "type": "pytorch-onnx_2.0",  "software_spec": {    "rev": "2",    "name": "runtime-23.1-py3.10"  } }</pre>																																																																												
Response Status Code	201	201																																																																												
Response Headers	<table><tr><td>Cache-Control</td><td>no-cache</td></tr><tr><td>Pragma</td><td>no-cache</td></tr><tr><td>Content-Length</td><td>783</td></tr><tr><td>Content-Type</td><td>application/json; charset=utf-8</td></tr><tr><td>Expires</td><td>-1</td></tr><tr><td>Location</td><td>https://management.azure.com/subscriptions/414c2448-40a3-428c-8a64-70a49b8415a7/resourceGroups/rogerresourcegroupthesis/providers/Microsoft.MachineLearningServices/workspaces/rogerworkspacethesis/models/testContainer?api-version=2024-04-01</td></tr><tr><td>x-ms-ratelimit-remaining-subscription-writes</td><td>199</td></tr><tr><td>Request-Context</td><td>appid-cid-v1.2d2e8e63-272e-4b3c-8596-4ee570a0e70d</td></tr><tr><td>x-ms-response-type</td><td>standard</td></tr><tr><td>Strict-Transport-Security</td><td>max-age=31536000; includeSubDomains</td></tr><tr><td>X-Content-Type-Options</td><td>nosniff</td></tr><tr><td>x-aml-cluster</td><td>vienna-eastus2-02</td></tr><tr><td>x-request-time</td><td>166</td></tr><tr><td>x-ms-ratelimit-remaining-subscription-global-writes</td><td>2999</td></tr><tr><td>x-ms-request-id</td><td>fee530ef-747c-4b99-a7d9-914f3bee3bd</td></tr><tr><td>x-ms-correlation-request-id</td><td>fee530ef-747c-4b99-a7d9-914f3bee3bd</td></tr><tr><td>x-ms-routing-request-id</td><td>WESTEUROPE.20240601T1933222.fee530ef-747c-4b99</td></tr><tr><td>X-Cache</td><td>CONFIG_NOCACHE</td></tr><tr><td>X-MSEdge-Ref</td><td>Ref A: 036801ED59C04E44AA40CD9E93B7F60B Ref B:</td></tr><tr><td>Date</td><td>Sat, 01 Jun 2024 19:33:22 GMT</td></tr></table>	Cache-Control	no-cache	Pragma	no-cache	Content-Length	783	Content-Type	application/json; charset=utf-8	Expires	-1	Location	https://management.azure.com/subscriptions/414c2448-40a3-428c-8a64-70a49b8415a7/resourceGroups/rogerresourcegroupthesis/providers/Microsoft.MachineLearningServices/workspaces/rogerworkspacethesis/models/testContainer?api-version=2024-04-01	x-ms-ratelimit-remaining-subscription-writes	199	Request-Context	appid-cid-v1.2d2e8e63-272e-4b3c-8596-4ee570a0e70d	x-ms-response-type	standard	Strict-Transport-Security	max-age=31536000; includeSubDomains	X-Content-Type-Options	nosniff	x-aml-cluster	vienna-eastus2-02	x-request-time	166	x-ms-ratelimit-remaining-subscription-global-writes	2999	x-ms-request-id	fee530ef-747c-4b99-a7d9-914f3bee3bd	x-ms-correlation-request-id	fee530ef-747c-4b99-a7d9-914f3bee3bd	x-ms-routing-request-id	WESTEUROPE.20240601T1933222.fee530ef-747c-4b99	X-Cache	CONFIG_NOCACHE	X-MSEdge-Ref	Ref A: 036801ED59C04E44AA40CD9E93B7F60B Ref B:	Date	Sat, 01 Jun 2024 19:33:22 GMT	<table><tr><td>Date</td><td>Sat, 08 Jun 2024 11:18:56 GMT</td></tr><tr><td>Content-Type</td><td>application/json</td></tr><tr><td>Content-Length</td><td>684</td></tr><tr><td>Connection</td><td>keep-alive</td></tr><tr><td>server-timing</td><td>InitId,desc=41d52eab7b4b5f3c</td></tr><tr><td>x-global-transaction-id</td><td>2e584e75d30dc509498a1843a57f15b1</td></tr><tr><td>Pragma</td><td>no-cache</td></tr><tr><td>Cache-Control</td><td>no-cache, no-store, must-revalidate</td></tr><tr><td>Content-Security-Policy</td><td>default-src 'none'; script-src 'self'; connect-src 'self'; img-src</td></tr><tr><td>X-Frame-Options</td><td>DENY</td></tr><tr><td>X-XSS-Protection</td><td>1; mode=block</td></tr><tr><td>X-Xss-Protection</td><td>1; mode=block</td></tr><tr><td>X-Content-Type-Options</td><td>nosniff</td></tr><tr><td>Location</td><td>/ml/v4/models/ea28244c-fbdf-4a1a-b555-247fec03d8a0</td></tr><tr><td>Strict-Transport-Security</td><td>max-age=31536000; includeSubDomains</td></tr><tr><td>CF-Cache-Status</td><td>DYNAMIC</td></tr><tr><td>Server</td><td>cloudflare</td></tr><tr><td>CF-RAY</td><td>89087be3eb20b98f-AMS</td></tr></table>	Date	Sat, 08 Jun 2024 11:18:56 GMT	Content-Type	application/json	Content-Length	684	Connection	keep-alive	server-timing	InitId,desc=41d52eab7b4b5f3c	x-global-transaction-id	2e584e75d30dc509498a1843a57f15b1	Pragma	no-cache	Cache-Control	no-cache, no-store, must-revalidate	Content-Security-Policy	default-src 'none'; script-src 'self'; connect-src 'self'; img-src	X-Frame-Options	DENY	X-XSS-Protection	1; mode=block	X-Xss-Protection	1; mode=block	X-Content-Type-Options	nosniff	Location	/ml/v4/models/ea28244c-fbdf-4a1a-b555-247fec03d8a0	Strict-Transport-Security	max-age=31536000; includeSubDomains	CF-Cache-Status	DYNAMIC	Server	cloudflare	CF-RAY	89087be3eb20b98f-AMS
Cache-Control	no-cache																																																																													
Pragma	no-cache																																																																													
Content-Length	783																																																																													
Content-Type	application/json; charset=utf-8																																																																													
Expires	-1																																																																													
Location	https://management.azure.com/subscriptions/414c2448-40a3-428c-8a64-70a49b8415a7/resourceGroups/rogerresourcegroupthesis/providers/Microsoft.MachineLearningServices/workspaces/rogerworkspacethesis/models/testContainer?api-version=2024-04-01																																																																													
x-ms-ratelimit-remaining-subscription-writes	199																																																																													
Request-Context	appid-cid-v1.2d2e8e63-272e-4b3c-8596-4ee570a0e70d																																																																													
x-ms-response-type	standard																																																																													
Strict-Transport-Security	max-age=31536000; includeSubDomains																																																																													
X-Content-Type-Options	nosniff																																																																													
x-aml-cluster	vienna-eastus2-02																																																																													
x-request-time	166																																																																													
x-ms-ratelimit-remaining-subscription-global-writes	2999																																																																													
x-ms-request-id	fee530ef-747c-4b99-a7d9-914f3bee3bd																																																																													
x-ms-correlation-request-id	fee530ef-747c-4b99-a7d9-914f3bee3bd																																																																													
x-ms-routing-request-id	WESTEUROPE.20240601T1933222.fee530ef-747c-4b99																																																																													
X-Cache	CONFIG_NOCACHE																																																																													
X-MSEdge-Ref	Ref A: 036801ED59C04E44AA40CD9E93B7F60B Ref B:																																																																													
Date	Sat, 01 Jun 2024 19:33:22 GMT																																																																													
Date	Sat, 08 Jun 2024 11:18:56 GMT																																																																													
Content-Type	application/json																																																																													
Content-Length	684																																																																													
Connection	keep-alive																																																																													
server-timing	InitId,desc=41d52eab7b4b5f3c																																																																													
x-global-transaction-id	2e584e75d30dc509498a1843a57f15b1																																																																													
Pragma	no-cache																																																																													
Cache-Control	no-cache, no-store, must-revalidate																																																																													
Content-Security-Policy	default-src 'none'; script-src 'self'; connect-src 'self'; img-src																																																																													
X-Frame-Options	DENY																																																																													
X-XSS-Protection	1; mode=block																																																																													
X-Xss-Protection	1; mode=block																																																																													
X-Content-Type-Options	nosniff																																																																													
Location	/ml/v4/models/ea28244c-fbdf-4a1a-b555-247fec03d8a0																																																																													
Strict-Transport-Security	max-age=31536000; includeSubDomains																																																																													
CF-Cache-Status	DYNAMIC																																																																													
Server	cloudflare																																																																													
CF-RAY	89087be3eb20b98f-AMS																																																																													
Response Body	<pre>{  "id": "/subscriptions/414c2448-40a3-428c-8a64-70a49b8415a7/resourceGroups/rogerresourcegroupthesis/providers/Microsoft.MachineLearningServices/workspaces/rogerworkspacethesis/models/testContainer",  "name": "testContainer",  "type": "Microsoft.MachineLearningServices/workspaces/models",  "properties": {    "description": "Model container description",    "tags": {},    "properties": {},    "isArchived": false,    "latestVersion": null,    "nextVersion": "1",    "provisioningState": "Succeeded"  },  "systemData": {    "createdAt": "2024-06-01T19:33:22.529952+00:00",    "createdBy": "Ruiz Salvat",    "createdByType": "User",    "lastModifiedAt": "2024-06-01T19:33:22.529952+00:00",    "lastModifiedBy": "Ruiz Salvat",    "lastModifiedByType": "User"  } }</pre>	<pre>{  "entity": {    "hybrid_pipeline_software_specs": [],    "software_spec": {      "id": "336b29df-e0e1-5e7d-b6a5-f6ab722625b2",      "name": "runtime-23.1-py3.10",      "rev": "2"    },    "type": "pytorch-onnx_2.0"  },  "metadata": {    "created_at": "2024-06-08T11:18:55.973Z",    "description": "This is my first resource.",    "id": "ea28244c-fbdf-4a1a-b555-247fec03d8a0",    "modified_at": "2024-06-08T11:18:55.973Z",    "name": "my-resource",    "owner": "IBMid-692000F6R5",    "project_id": "cce6c1cd-30e5-4b16-a380-966592b1a296",    "resource_key": "28f8c7c-50e0-41fd-8b3c-8596-4ee570a0e70d",    ...  } }</pre>																																																																												

Figure 8: Comparison of two HTTP requests for creating an ML model resource

HTTP Responses from Bad Requests			
API	G. Vertex AI	BigML	IBM ML
Endpoint	Export dataset	Data source create	Pipeline update
Response Status Code	500	400	400
Response Body	<pre>{   "error": {     "code": 500,     "message": "Internal error encountered.",     "status": "INTERNAL"   } }</pre>	<pre>{   "code": 400,   "status": {     "code": -1204,     "message": "Bad request"   } }</pre>	<pre>{   "trace": "86640ecf2747438181a09097bffd0730",   "errors": [     {       "code": "invalid_request_entity",       "message": "Invalid request entity: Patch path 'name' not allowed.",       "more_info": "https://cloud.ibm.com/apidocs/machine-learning"     }   ],   "status_code": "400" }</pre>

Figure 9: HTTP responses to malformed requests

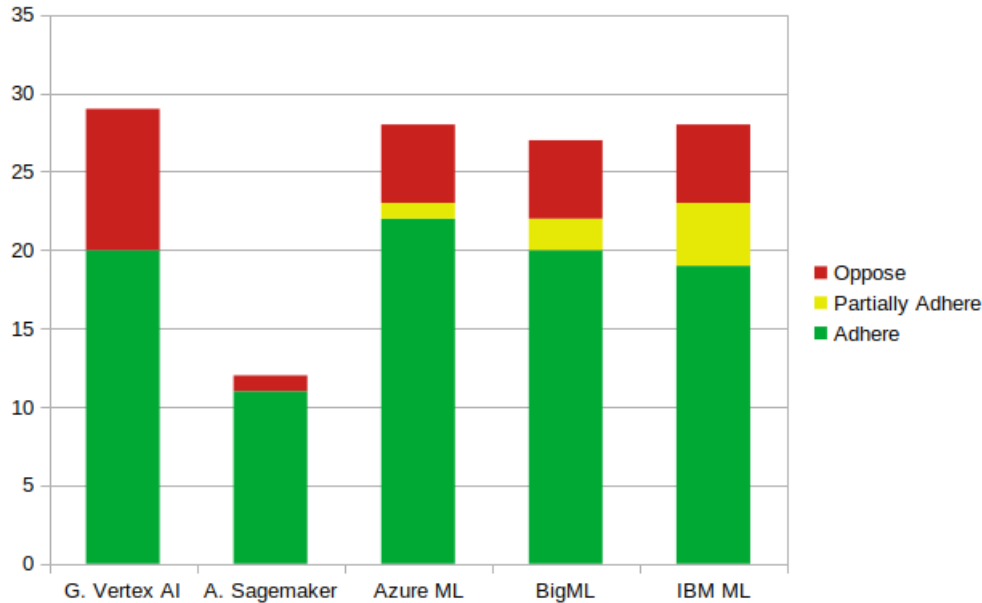


Figure 10: REST rule compliance per ML API

of web APIs. First, it is unclear if all rule violations really impact the maturity of APIs. Secondly, the maturity model might require a greater level of granularity per level. For example, when assessing the use of hypermedia, it is possible that an API implements it well in the response body payloads, but does not implement it in response metadata.

Additionally, from the thematic analysis it is noticeable that most of the providers offer API accessibility through client libraries or client SDKs. Depending on the provider there are more or less interfaces available. In particular, AWS offers a wide range of SDKs that developers can install as packages to their applications. However, AWS does not provide HTTP methods that can be directly accessed by clients. Instead, the SDKs serve as wrappers for users to access

ML functionalities through software components but not through HTTP requests.

For future research, it would be valuable to discover the rationale behind the choice for ML platform to package ML APIs and promote the use of SDKs. Several concepts that emerged during thematic analysis could be related with this rationale. First, we have observed a relevance for the specification of API versions. Cloud SDKs, could be somehow more convenient to manage for ML platforms to ensure client compatibility as packaged APIs can be installed through dependency managers. Another related concept is a tendency for developing containerized applications and deploying containerized services. Communicating with the cloud infrastructure through



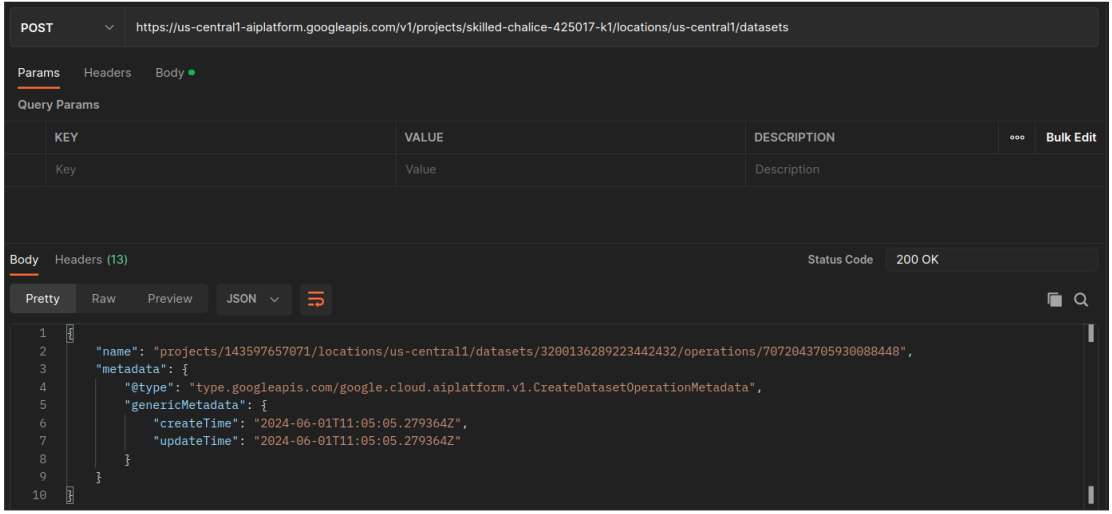
Violation rule 28: 201 ("Created") must be used to indicate successful resource creation	
API	G. Vertex ai
Expected status code	201
Received status code	200
Postman request example	 <pre> 1  POST https://us-central1-aiplatform.googleapis.com/v1/projects/skilled-chalice-425017-k1/locations/us-central1/datasets 2 3  Params Headers Body 4  Query Params 5  KEY VALUE DESCRIPTION Bulk Edit 6  Key Value Description 7 8  Body Headers (13) Status Code 200 OK 9  Pretty Raw Preview JSON 10 11  1 "name": "projects/143597657071/locations/us-central1/datasets/3200136289223442432/operations/7072043705930088448", 12  2 "metadata": { 13  3   "type": "type.googleapis.com/google.cloud.aiplatform.v1.CreateDatasetOperationMetadata", 14  4   "genericMetadata": { 15  5     "createTime": "2024-06-01T11:05:05.279364Z", 16  6     "updateTime": "2024-06-01T11:05:05.279364Z" 17  7   } 18  8 } 19  9 20  10 </pre>

Figure 11: Example violation of rule 28

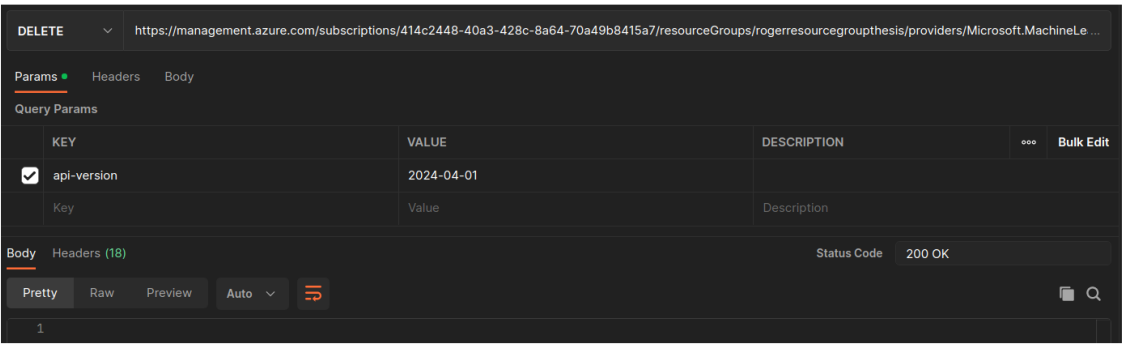
Violation rule 30: 204 ("No Content") should be used when the response body is intentionally empty	
API	Azure ML
Expected status code	204
Received status code	200
Postman request example	 <pre> 1  DELETE https://management.azure.com/subscriptions/414c2448-40a3-428c-8a64-70a49b8415a7/resourceGroups/rogerresourcegroupthesis/providers/Microsoft.MachineLe... 2 3  Params Headers Body 4  Query Params 5  KEY VALUE DESCRIPTION Bulk Edit 6  [x] api-version 2024-04-01 7  Key Value Description 8 9  Body Headers (18) Status Code 200 OK 10  Pretty Raw Preview Auto 11 12  1 </pre>

Figure 12: Example violation of rule 30

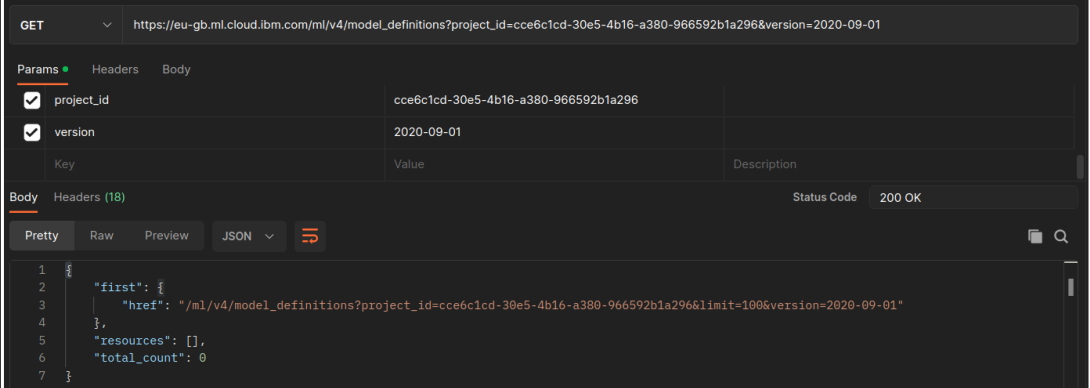
cloud SDKs could facilitate the configuration of network settings for the containerized image.

## 6 THREATS TO VALIDITY

Several threats to validity have been considered according to Wohlin et al. [27]. However, as a qualitative study, some of the considerations are approached differently. We used survey questions as proposed by Feldt et al. [9] to find out possible threats to validity for indicating the dimensions derived from Wohlin's et al. [27] work.

### 6.1 Internal Validity

RQ1 is an inductive analysis based on principles of qualitative coding. The main criticism towards research methods based on coding techniques such as grounded theory, which shares a lot of similarities with the thematic analysis applied in this study, is the lack of rigor caused by the subjectivism introduced by the researcher. While internal validity is often related with mathematical proofs and achieving statistical significance, interpretivist approaches like thematic analysis must ensure internal consistency in order to have

Violation rule 4: Underscores ( _ ) should not be used in URI	
API	IBM ML
Additional remarks	Note the use of hypermedia in the response body
Postman request example	 <pre> 1  { 2    "first": { 3      "href": "/ml/v4/model_definitions?project_id=cce6c1cd-30e5-4b16-a380-966592b1a296&amp;limit=100&amp;version=2020-09-01" 4    }, 5    "resources": [], 6    "total_count": 0 7  } </pre>

**Figure 13: Example violation of rule 4**

all parts of the theory fit with each other explaining the collected data [11].

Although this limitation is mitigated when doing the constant comparison between emerging theory and new coded data, several aspects of the analysis still rely on subjective assumptions of the researcher. For instance, at the open coding phase, the selection of documents might already introduce bias as the researcher might not know at this point what literature is relevant. Contrary, researchers might also attempt to code too much data resulting in a data overload. Another point of subjectivity is when judging theoretical saturation, as some aspects of the domain might still need to be discovered [11].

Nevertheless, in this study, two researchers collaborated in the coding phase for contrasting, a priori, coded annotations from data and the emerging categories. The selection of endpoints remains subjective because the researchers did not make a prior agreement on which documents needed to be downloaded. However, this was not considered a major threat because only the API documentation from the official platform websites was used. Having two researchers validate the coding phase impacted the codes, as the common data points could be emphasized later in the research, and the uncommon ones excluded from the analysis. Despite the effort to achieve impartial results, some risks of misinterpretation remain from the coding phase, as both researchers had a similar level of expertise in IS research, and it is still a small group.

## 6.2 External Validity

Gasson [11] suggests that, in order to achieve external validity when merging codes into categories, in inductive research the emerging theory must be transferable to other objects. For example, a certain characteristic of APIs that emerged from axial coding such as API versioning, must also be applicable to other APIs that were not considered in this study.

Nevertheless, it is important to note that this transferability does not imply how such a characteristic is found in the study object e.g. if the API version is specified in the URI path or in the query parameters. Instead, transferability refers to the fact

that such a characteristic is somehow considered when analyzing the study object. An example of a non-transferable characteristic which should not be part of the generated theory could be how APIs implement the OAuth2 protocol, as not all APIs might implement it e.g. BigML. Alternatively, a valid characteristic could be how APIs implement authentication protocols, as this can be considered by all APIs.

A powerful use case in which the results and conclusions of this study could be generalized to other APIs is the creation of design guidelines for web APIs tailored to MLaaS platforms. Such guidelines could exist for both developers incorporating ML services into their products and API designers from cloud platforms aiming to provide reusable and maintainable solutions. Several concepts discussed in this study could already be part of design guidelines, such as API versioning and including hypermedia in HTTP responses for linking data entities. Other concepts, like handling forbidden access requests or the body format for updating resources, may still need to be further discussed to find the best possible approach.

## 6.3 Construct Validity

Construct validity refers to how well a research method measures the specific concept it is intended to capture. In thematic analysis, this means ensuring the identified themes accurately reflect the aspects of the data we aimed to explore. As an iterative process, we started by collecting data from a broader range of documents to get a general understanding. Then, we refined the selection of documents by narrowing down to more specific documents like REST references from MLaaS APIs.

For the manual analysis in RQ2, one potential source of bias in assessing compliance could be the interpretation of the rules. Some rules include the term "should," which makes it unclear how strictly it should be enforced. To mitigate this lack of clarity, we established a priori (see section 3.3) how adherence to the REST rules would be determined.

## 6.4 Conclusion Validity

Unlike RQ1, RQ2 relies on a deductive approach, where existing theory is tested against data. In RQ2, the main concern regarding validity is whether the selected set of endpoints is representative for analyzing REST compliance of APIs. In other words, there's a potential for missing relevant rules or including irrelevant ones. However, we believe the risk of an unrepresentative set is mitigated by the selection process, which considers the importance of REST rules outlined by kotstein et al. [15].

## 7 CONCLUSION

Thematic analysis on ML cloud APIs revealed the core components of an MLaaS platforms and how they interact with each other. This higher-level conceptualization helped identify which elements relate with REST APIs within the context of MLaaS platforms.

Inference APIs can be set up for exposing deployed ML models that can make predictions on input data. On the other hand, REST APIs can also be used to manage resources from the MLaaS platform. Moreover, in MLaaS platforms, resources tend to get organized by projects or workspaces, and there is a clear need for ML pipelines for orchestrating all the processes required to build and deploy ML models. This is encompassed in a set of functionalities called MLOps.

The design of REST APIs on MLaaS platforms is influenced by the need for managing these ML Operations, as they are often dependent on each other. For instance, it is not possible to deploy an ML model if it has not been trained before. Therefore, achieving the level 3 of maturity (hypermedia controls) from Richardson's levels of maturity becomes more relevant, as users need a way for consulting the state of ML Resources e.g. for checking the state of an ML model that is being trained.

Furthermore, some ML models can be computationally expensive or have significant latency during predictions. This necessitates process scheduling, allowing developers to retrieve the state of their models. However, only the larger platforms, AWS and Google Cloud, provide links within HTTP responses to access model metadata and prediction outputs. Conversely, all APIs except Azure ML offer links pointing to data sources.

Variations were also encountered in API version specification within HTTP requests. The format used to specify the API version, as well as its data payload within an HTTP request, can vary depending on the API. Notably, API versioning is a critical consideration for MLaaS platforms due to the rapid evolution of ML technologies and the frequent updates they necessitate for cloud infrastructure. All of the studied APIs enhance the specification of the API version in HTTP requests, and it is required by almost all of them.

Moreover, discrepancies were found on the design of resource update services. From the use of HTTP verbs, where some APIs use PATCH for partial updates, up to the body payload format required in HTTP updates, where the IBM cloud API implements it in a JsonPatchOperation format. Other discrepancies were found on the specification of secret keys for authentication, the specification of the server location, and the use of query parameters.

Additionally, our study revealed differences in error handling practices among the APIs. While attempts at unauthorized access

seem to be correctly handled and implemented with standardized responses, forbidden access and malformed requests are responded to differently by the APIs. We have seen different levels of detail in the information about the error provided in both the HTTP status codes and bodies. A 500 Internal Server Error response might hinder debuggability while obfuscating potential API vulnerabilities in malformed requests. Similarly, a 404 error could lower the understandability of the API while protecting against potential API forbidden access requests.

The analysis of REST compliance of the selected ML cloud APIs showed that GET and DELETE verbs are generally used correctly. Similarly, nonspecific success and unauthorized code response codes are returned when they are expected to be returned. Moreover, the content type is always specified and JSON always used as a payload format. Nevertheless, some APIs lack adherence to other REST rules, such as specifying content length and resource location in response metadata. Additionally, partial compliance is observed with other rules. For instance, empty responses may not always use the appropriate 204 status code, and the POST verb may not be consistently used for executing controllers. Notably, enforcing catching in responses seems to be less prioritized for ML APIs.

Finally, while overall compliance is relatively high (with 74.19% of the rules fully followed), specific violations, such as inconsistent status code usage and bad URI design, can impact API usability and interoperability. While Azure demonstrates the highest compliance, IBM exhibits the most deviations.

## REFERENCES

- [1] VV Arutyunov. 2012. Cloud computing: Its history of development, modern state, and future considerations. *Scientific and Technical Information Processing* 39 (2012), 173–178.
- [2] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. 2020. Checking security properties of cloud service REST APIs. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 387–397.
- [3] Justus Bogner, Sebastian Kotstein, Daniel Abajirov, Timothy Ernst, and Manuel Merkel. 2024. RESTruler: Towards Automatically Identifying Violations of RESTful Design Rules in Web APIs. *arXiv preprint arXiv:2402.13710* (2024).
- [4] Justus Bogner, Sebastian Kotstein, and Timo Pfaff. 2023. Do RESTful API design rules have an impact on the understandability of Web APIs? *Empirical software engineering* 28, 6 (2023), 132.
- [5] Atakan Cetinsoy, Francisco J Martin, José Antonio Ortega, and Poul Petersen. 2016. The past, present, and future of machine learning APIs. In *Conference on Predictive APIs and Apps*. PMLR, 43–49.
- [6] Michael Coblenz, Wentao Guo, Kamatchi Voozhian, and Jeffrey S Foster. 2023. A Qualitative Study of REST API Design and Specification Practices. In *2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 148–156.
- [7] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*. IEEE, 275–284.
- [8] Frederico Duro, Jose Fernando S Carvalho, Anderson Fonseca, and Vinicius Cardoso Garcia. 2014. A systematic review on cloud computing. *The Journal of Supercomputing* 68 (2014), 1321–1346.
- [9] Robert Feldt and Ana Magazinius. 2010. Validity threats in empirical software engineering research—an initial survey. In *Seke*. 374–379.
- [10] M Fowler. 2010. Richardson maturity model: steps toward the glory of rest. <https://martinfowler.com/articles/richardsonMaturityModel.html>. [Accessed 16-07-2024].
- [11] Susan Gasson. 2004. Rigor in grounded theory research: An interpretive perspective on generating theory from qualitative field studies. In *The handbook of information systems research*. IGI Global, 79–102.
- [12] Rafia Islam, Vardhan Patamsetti, Aparna Gadhi, Ragha Madhavi Gondu, Chinna Manikanta Bandaru, Sai Chaitanya Kesani, and Olatunde Abiona. 2023. The future of cloud computing: benefits and challenges. *International Journal of Communications, Network and System Sciences* 16, 4 (2023), 53–65.

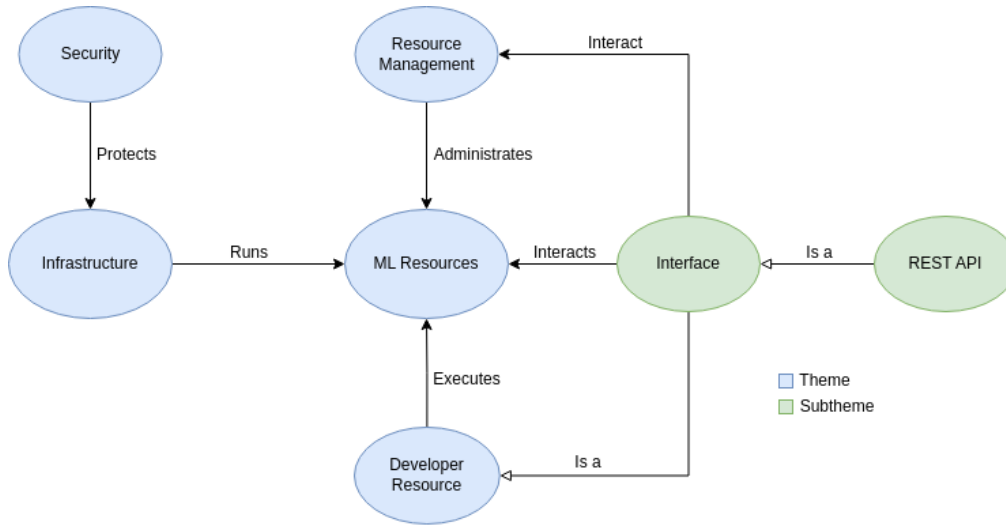
- [13] Can Kaymakci, Simon Wenninger, Philipp Pelger, and Alexander Sauer. 2022. A systematic selection process of machine learning cloud services for manufacturing SMEs. *Computers* 11, 1 (2022), 14.
- [14] Pramod P Khargonekar and Munther A Dahleh. 2018. Advancing systems and control research in the era of ML and AI. *Annual Reviews in Control* 45 (2018), 1–4.
- [15] Sebastian Kotstein and Justus Bogner. 2021. Which restful api design rules are important and how do they improve software quality? a delphi study with industry experts. In *Service-Oriented Computing: 15th Symposium and Summer School, SummerSOC 2021, Virtual Event, September 13–17, 2021, Proceedings 15*. Springer, 154–173.
- [16] Mark Masse. 2011. *REST API design rulebook*. " O'Reilly Media, Inc".
- [17] Naoel Moha, Francis Palma, Mathieu Nayrolles, Benjamin Joyen Conseil, Yann-Gaël Guéhéneuc, Benoît Baudry, and Jean-Marc Jézéquel. 2012. Specification and detection of SOA antipatterns. In *Service-Oriented Computing: 10th International Conference, ICSOC 2012, Shanghai, China, November 12–15, 2012. Proceedings 10*. Springer, 1–16.
- [18] Muhammad Naeem, Wilson Ozuem, Kerry Howell, and Silvia Ranfagni. 2023. A step-by-step process of thematic analysis to develop a conceptual model in qualitative research. *International Journal of Qualitative Methods* 22 (2023), 16094069231205789.
- [19] Andy Neumann, Nuno Laranjeiro, and Jorge Bernardino. 2018. An analysis of public REST web service APIs. *IEEE Transactions on Services Computing* 14, 4 (2018), 957–970.
- [20] Assel Omarova, Fazli Shermatov, and Gabriela Fuentes. 2024. Machine Learning Tech Giants: What are the Current Trends in Research? *Available at SSRN 4792516* (2024).
- [21] Francis Palma, Johann Dubois, Naoel Moha, and Yann-Gaël Guéhéneuc. 2014. Detection of REST patterns and antipatterns: a heuristics-based approach. In *Service-Oriented Computing: 12th International Conference, ICSOC 2014, Paris, France, November 3–6, 2014. Proceedings 12*. Springer, 230–244.
- [22] Fabio Petrillo, Philippe Merle, Naoel Moha, and Yann-Gaël Guéhéneuc. 2016. Are REST APIs for cloud computing well-designed? An exploratory study. In *Service-Oriented Computing: 14th International Conference, ICSOC 2016, Banff, AB, Canada, October 10–13, 2016. Proceedings 14*. Springer, 157–170.
- [23] Adnan Qayyum, Aneeqa Ijaz, Muhammad Usama, Waleed Iqbal, Junaid Qadir, Yehia Elkhathib, and Ala Al-Fuqaha. 2020. Securing machine learning in the cloud: A systematic review of cloud machine learning security. *Frontiers in big Data* 3 (2020), 587139.
- [24] Carlos Rodríguez, Marcos Baez, Florian Daniel, Fabio Casati, Juan Carlos Trabucco, Luigi Canali, and Gianraffaele Percannella. 2016. REST APIs: A large-scale analysis of compliance with principles and best practices. In *Web Engineering: 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6–9, 2016. Proceedings 16*. Springer, 21–39.
- [25] Iqbal H Sarker. 2021. Machine learning: Algorithms, real-world applications and research directions. *SN computer science* 2, 3 (2021), 160.
- [26] Michael Williams and Tami Moser. 2019. The art of coding and thematic exploration in qualitative research. *International management review* 15, 1 (2019), 45–55.
- [27] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, International Series in Software Engineering.
- [28] Qi Zhang, Lu Cheng, and Raouf Boutaba. 2010. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications* 1 (2010), 7–18.

## A APPENDIX A: HIGHER ORDER DIAGRAM

Figure 14 illustrates the relationships between various concepts in the context of ML Cloud APIs. ML Resources are the core assets provided by the cloud platform, encompassing ML models, algorithms, and datasets that developers can utilize. ML Resources are executed upon the Infrastructure, meaning the cloud platform’s infrastructure provides the computational power and environment necessary to run ML models and algorithms. Interaction between various components within the system is highlighted, such as developers interacting with the ML resources through the interface, and the interface interacting with the ML resources to execute tasks. The REST API is a specific type of interface that adheres to the REST architectural style, enabling communication between different software applications and the ML platforms using HTTP requests.

## B APPENDIX B: CODE DEFINITIONS

Table 5 presents a glossary of terms essential to understanding Machine Learning as a Service (MLaaS). It encompasses definitions for core MLaaS components such as interfaces, models, and datasets; ML processes like inference and its variations; development tools including SDKs and API management; communication protocols like HTTP methods and headers; and architectural concepts like hypermedia and data linking.



**Figure 14: Higher order diagram of themes**

**Table 5: Code definitions**

Code	Definition
MLaaS	Cloud computing platform as Service with ML products and functionalities
Interfaces	Computer software that allows developers to interact with MLaaS
REST API	Interface in a form of HTTP methods which adhere to REST best practices
ML Model	Computer program specifically designed to make predictions by leveraging statistical methods and analyzing data
Dataset	Collection of data points used to train and evaluate ML models
Data Source	Online location from where a dataset can be retrieved
Inference	Trained ML model accessible for making predictions
Real-time Inference	Inference that can be used for low-latency prediction requests
Batch Inference	Inference that can be used for high-latency, computer-demanding, prediction requests
Cloud SDK	Packaged set of development tools that abstracts the complexity of the underlying API
API Compatibility	Degree to which an API can interact with systems without encountering unexpected changes or breaking functionality
API Versioning	Practice of managing changes to an API by introducing distinct versions to ensure API compatibility
HTTP Verbs	Specific keywords that define the desired action to be performed on a particular resource
HTTP Headers	Information included in both HTTP requests and responses which provides additional details about the request or response beyond the core data
Hypermedia	Controls that tells clients what actions to do next, and the URI of the resource needed to manipulate, rather than having the client to know where to post the request
Data Entity Link	Resource link included in an HTTP response that points to a data source
HATEOAS	Principle that emphasizes using hypermedia links within responses to guide clients through available actions and resources