GraphQL: A Systematic Mapping Study

ANTONIO QUIÑA-MERA, FICA Faculty, eCIER Research Group, Universidad Técnica del Norte, Ecuador and SCORE Lab, Universidad de Sevilla, Spain
PABLO FERNANDEZ, JOSÉ MARÍA GARCÍA, and ANTONIO RUIZ-CORTÉS, SCORE Lab, I3US Institute, Universidad de Sevilla, Spain

GraphQL is a query language and execution engine for web application programming interfaces (APIs) proposed as an alternative to improve data access problems and versioning of representational state transfer APIs. In this article, we thoroughly study the GraphQL field, first describing the GraphQL paradigm and its conceptual framework, and then conducting a systematic mapping study of 84 primary studies selected from an original set of 3,185. Our work analyzes trends or knowledge gaps about GraphQL by general classification of the studies and specific classification of this research topic. The study's main conclusions show that GraphQL adoption is growing in the community as a strong alternative to implement APIs. However, we identified the need to strengthen the amount and rigor of empirical evidence collection in applied industry and government studies. In addition, we revealed the opportunity for specific studies on most GraphQL components, especially the consumption of GraphQL API services.

CCS Concepts: • **Applied computing** → **Service-oriented architectures**; • **Software and its engineering** → **Software as a service orchestration system**; **API languages**; • **Computer systems organization** → *Cloud computing*; • **Information systems** → *RESTful web services;*

Additional Key Words and Phrases: GraphQL, API, microservices, systematic mapping study

ACM Reference format:
Antonio Quiña-Mera, Pablo Fernandez, José María García, and Antonio Ruiz-Cortés. 2023. GraphQL: A Systematic Mapping Study. *ACM Comput. Surv.* 55, 10, Article 202 (February 2023), 35 pages.
https://doi.org/10.1145/3561818

1 INTRODUCTION

Today, **Software-as-a-Service (SaaS)** has received significant attention as one of the three main service models of cloud computing (i.e., **Platform-as-a-Service or (PaaS)** and **Infrastructure-as-a-Service or (IaaS)** [33, 54]). SaaS utilizes the internet to deliver applications to its users, which are managed by a provider. Therefore, selecting the best SaaS provider among those available is

This work is partially supported by the Universidad Técnica del Norte (UTN), Ecuador and Grants No. RTI2018-101204-B-C21, No. RTI2018-101204-B-C22, No. PID2021-126227NB-C21, and No. PID2021-126227NB-C22, funded by MCIN/AEI/10.13039/501100011033/ and "ERDF a way of making Europe," and Grants No. PYC20 RE 084-US, No. P18-FR-2895, No. US-1264651, and No. US-1381595, funded by Junta de Andalucia/ERDF, UE.
Authors' addresses: A. Quiña-Mera (corresponding author), FICA Faculty, eCIER Research Group, Universidad Técnica del Norte, Av. 17 de Julio 5-21, Ecuador, Imbabura, Ibarra, 100105 and SCORE Lab, Universidad de Sevilla, Av. Reina Mercedes s/n, 41012, Sevilla, Spain; email: aquina@utn.edu.ec; P. Fernandez, J. M. García, and A. Ruiz-Cortés, SCORE Lab, I3US Institute, Universidad de Sevilla, Av. Reina Mercedes s/n, 41012, Sevilla, Spain; emails: {pablofm, josemgarcia, aruiz}@us.es.

0360-0300/2023/02-ART202
https://doi.org/10.1145/3561818

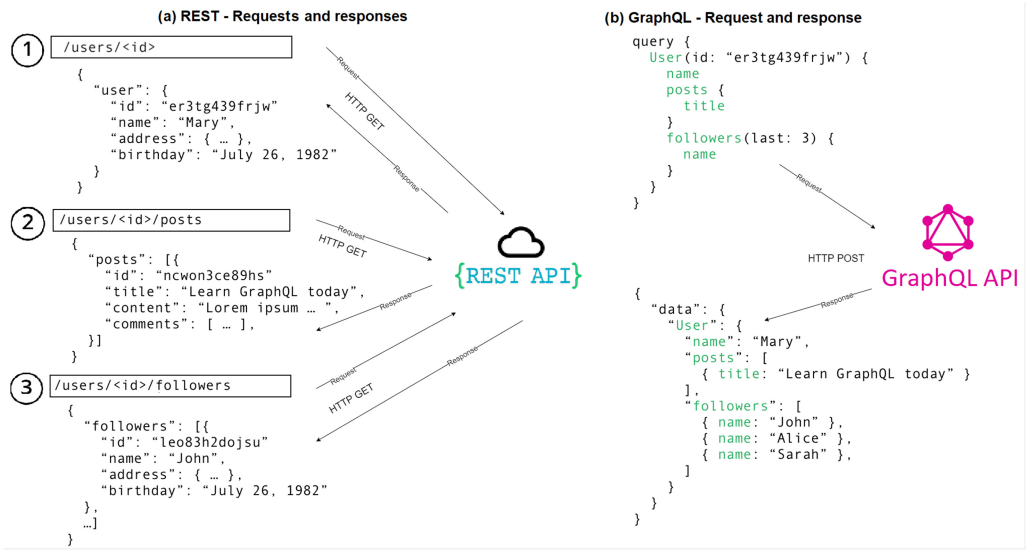


Fig. 1. Example of over-fetching and under-fetching in REST API vs. GraphQL API [44].

a critical challenge [46]. This challenge falls on software architects who must build reliable and efficient SaaS that overcome the significant technical and functional problems of cloud applications, such as multi-tenancy, redundancy, recovery, or scalability [54]. To overcome these challenges, the microservices architecture represents a clear trend both in academia and industry [29], where large companies worldwide have evolved their applications towards this architectural style [8].

Specifically, the **Microservices Architecture (MSA)** is composed of small applications (microservices) with a single responsibility (functional, non-functional, or cross-functional requirements) that can be independently deployed, scaled, and tested [2, 53]. Microservices can be developed and modified independently using different programming languages and product stacks, thus supporting agility [2, 23]. Its main advantages are maintainability, reusability, scalability, availability, and automated deployment [29]. Microservices communicate through lightweight mechanisms, typically using the **Representational State Transfer (REST)** architectural style to build **application programming interfaces (APIs)** named REST APIs. Software organizations have rapidly adopted REST APIs in the development of their applications after its creation in 2000 [56].

REST is the most widely used paradigm in microservices development [61]. However, despite its popularity, it presents some data access issues, such as: (i) query complexity (i.e., REST requires multiple HTTP requests to obtain multiple resources); (ii) over-fetching (i.e., a REST request returns more data than an application needs); and (iii) under-fetching (i.e., a particular endpoint does not give enough information, so additional requests must be made to obtain the required information), which is also known in the literature as the n+1 request problem [34, 55]. To illustrate the over-fetching and under-fetching problems found in REST microservices, we consider a scenario of consuming a REST API and a GraphQL API of blogs [44]. In this scenario, a client application needs to query post titles of a specific user and the names of the last three followers of that user, see Figure 1.

On the one hand, Figure 1(a) shows the request and response to the REST API. REST usually collects the data by accessing various resources (aka endpoints). For the proposed scenario, it is necessary to query data at three endpoints: `/users/<id>` to get the user's data, second, `/users/<id>/posts` for the user's posts, and the third endpoint `/users/<id>/followers`, which

returns a list of followers per user. On the other hand, Figure 1(b) shows the request and response to the GraphQL API. Using this paradigm, the user can send only one query asking for the specific data it needs, then the server responds only with the requested data [44, 55]. The under-fetching in REST surfaces when the request needs to access three endpoints to get the data requirements it needs, while GraphQL needs only one access to get the required data. In turn, over-fetching in REST occurs when each endpoint access gets more data than it needs, while GraphQL only gets the requested data.

In the past decade, some query languages such as SPARQL,[1] Cypher,[2] Gremlin,[3] and GraphQL have emerged as alternatives for API data access [50]. In this study, we focus on GraphQL, because there is a growing interest in the industry since it has proven to be an alternative to solve the problems found in traditional REST technology [50, 55]. GraphQL started in 2012 as an internally developed specification on Facebook. In 2015, they decided to share GraphQL with a broader community [28], releasing both an open source specification and a reference implementation through the graphql.org[4] community [12]. Since then, the community has grown, as well as the adoption in the production environment by hundreds of organizations of all sizes, such as Atlassian, GitHub, Netflix, The New York Times, and Twitter, among others [28, 36]. In this work, we evidence an average annual growth of 16.67% between 2015 and 2020 in the number of studies published in scientific literature databases related to software engineering. In the same sense, we show an average annual growth of 13.61% in the search trend of the term "graphql," reported in Google Trends (see Section 3.1). Despite the growing interest in GraphQL, there is no global vision that helps new researchers know the principal existing research proposals, the studies' scope, trends in the research approach over time, or existing gaps in scientific evidence.

This work's objective is twofold: first, to introduce the novel GraphQL paradigm through illustration, exemplification, and conceptualization of its components. Then, to conduct a **systematic mapping study (SMS)** of GraphQL to establish an overview of the subject through a publication classification scheme and structure the scientific community's field of interest. The rest of the article is structured as follows. In Section 2, we establish the GraphQL paradigm. Next, Section 3 shows how we conducted the SMS. Then, in Section 4, we present the results obtained from the SMS. Afterwards, we analyze and discuss the main findings of our study. Finally, Section 6 summarizes the conclusions of the study.

2 GRAPHQL PARADIGM

2.1 GraphQL Foundations

Its formal specification defines GraphQL as a query language and execution engine to describe the capabilities and requirements of data models for client-server applications [52]. To implement **GraphQL application servers (GraphQL APIs)**, it is not necessary to use a specific programming language or persistence mechanism. Instead, GraphQL encodes in a uniform language the model capabilities of a type system based on five design principles: hierarchical, product-centric, strong-typing, client-specified queries, and introspective [12, 52].

In the following, we present the GraphQL paradigm, illustrating the conceptual and functional structure of the GraphQL formal specification[5] (June 2018 release). We begin to structure the paradigm by showing the interaction between the high-level components of GraphQL (language

[1]See https://www.w3.org/TR/rdf-sparql-query.
[2]See https://neo4j.com/developer/cypher.
[3]See https://tinkerpop.apache.org/gremlin.html.
[4]See http://graphql.org.
[5]See https://graphql.github.io/graphql-spec.

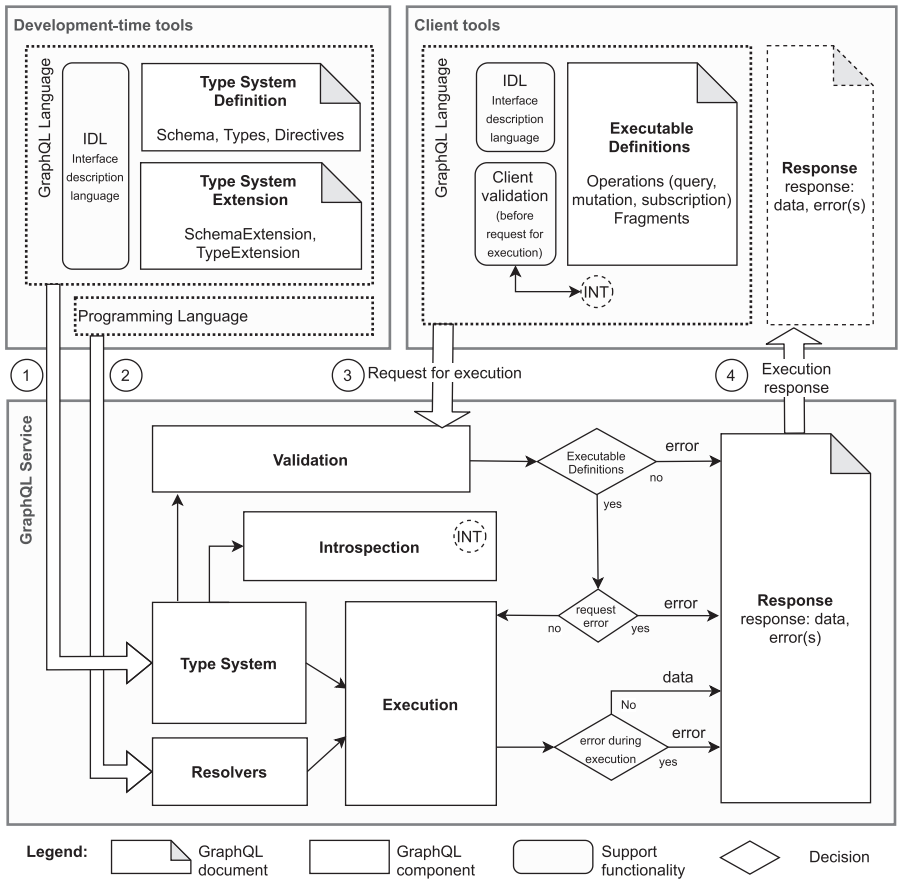


Fig. 2. Principal components interaction of the GraphQL paradigm.

and its grammar, type system, introspection, and execution and validation engines). Figure 2 shows that the interaction starts in (1) the development-time tools that use the GraphQL language, its grammar, and **Interface Description Language (IDL)** to define and generate the type system or type system extensions of the GraphQL Service. (2) The generation of the service (GraphQL API) is complemented by generating type system Resolvers using some programming language. Then the service generated from the customer tools is consumed. (3) A document with executable operation definitions (query, mutation, subscription) or fragments is created to send an execution request to the GraphQL service. Before sending the request, the client tools provide a syntax validation using the GraphQL service's introspection (represented with the "INT" bubble in the "Client tools" section). The GraphQL service receives the request and validates that the document contains only executable definitions for execution. Running the GraphQL service obtains the requested data from the Type System and Resolvers. (4) The GraphQL service execution sends the result as a response document (usually in JSON format) to the client tools. The response document also includes error messages if there are errors during execution.

Once explained how the four high-level components work and interact, in the next Section 2.2, we complete their structure and technical description of 17 specific components of the GraphQL paradigm.

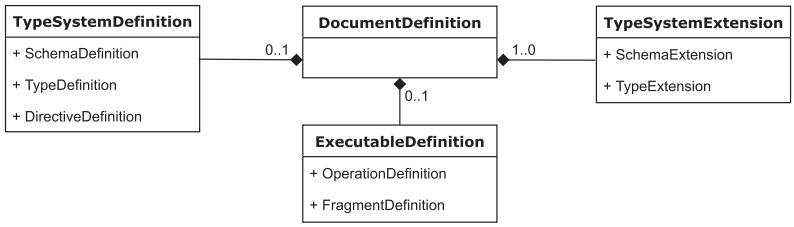


Fig. 3. GraphQL document definition.

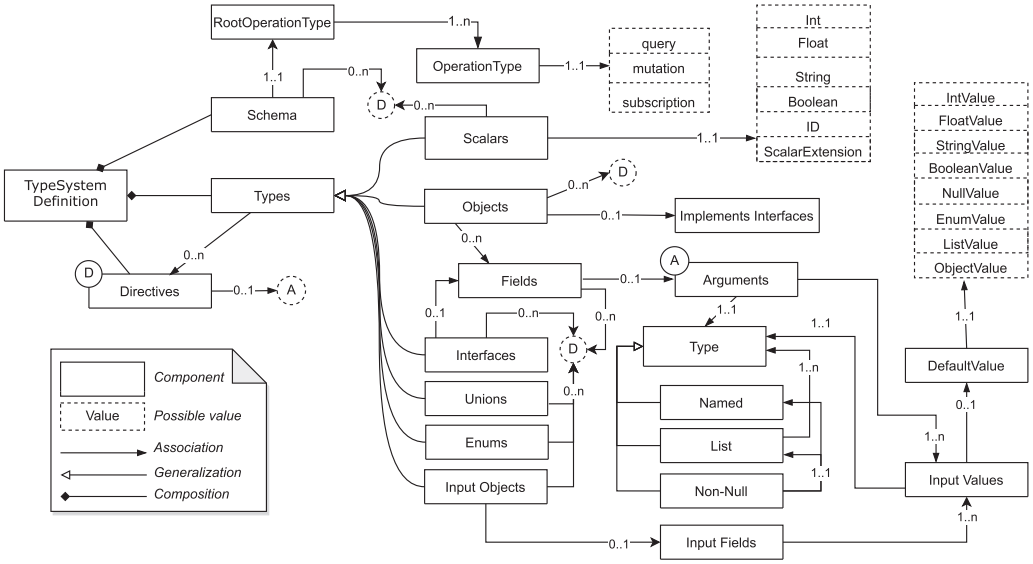


Fig. 4. GraphQL type system definition.

2.2 Operational Semantics

2.2.1 GraphQL Language. The GraphQL language provides a comprehensive syntax to create documents that can contain either the type system definitions of GraphQL services or the actual queries clients execute on them. Thus, a GraphQL document can contain different definition instances (type system, type system extension, or executable) [52]. Figure 3 shows each type of definition's main components; they will be detailed and exemplified in the following.

Besides, GraphQL language includes an IDL used to describe the type system of a GraphQL service used by the tools to provide utilities such as client code generation or service bootstrapping [52].

Type System Definition. Defines the data capabilities of the *Type System* of a GraphQL server, as well as the input types of the query variables. A GraphQL Document containing a *TypeSystemDefinition* must not be executed by GraphQL execution services. We illustrate the conceptual structure and the interaction of the components of the type system definition in Figure 4. The main components are described in the following.

Schema conforms the collective-type system capabilities of a GraphQL service. Their definition is in terms of types and directives, as well as the root operation types for each kind of operation: query, mutation, and subscription.

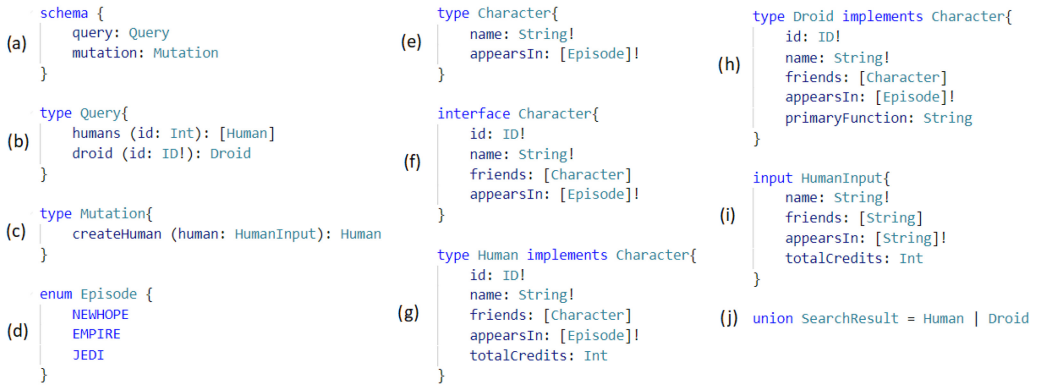


Fig. 5. GraphQL type system example.

Types are the fundamental unit of the GraphQL schema. Concrete types can be defined based on six named types and two wrapping types. The "named types" are:

- Scalars: represent a primitive value, like string, integer, float, Boolean, or a unique **identifier (ID)**.
- Enums: describe the set of possible values.
- Objects: define a set of fields where each field is also of a type in the system. Fields are conceptually functions that return values and occasionally accept arguments that alter their behavior; these arguments are usually mapped directly to a function within a GraphQL server implementation. Therefore, response trees are composed of leaves of type Scalars and Enums, and intermediate levels are of type Objects, thus allowing arbitrary type hierarchies to be defined.
- Input objects: defines a set of input fields; it can be scalars, enums, or other input objects.

In turn, among "named types" there are two abstract types that must first resolve to a relevant "named type" object:

- Interfaces: define a list of fields.
- Unions: define a list of possible types.

Finally, we refer to wrapping types as those that effectively wrap or contain a "named type":

- List: a GraphQL schema may describe that a field represents a list of other types; for this reason, the list type wraps another type.
- Non-null: wraps another type and denotes that the resulting value will never be null.

Directives provide a way to describe alternate runtime execution and type validation behavior in a GraphQL document. Directives can be used to describe additional information for types, fields, fragments, and operations.

Next, we complement the conceptualization of the GraphQL paradigm specification with illustrations and examples of its components. For this purpose, we implement a Star Wars API GraphQL service that shows the movies' basic structure and characters.

Figure 5 shows some examples containing the essential components of a type system definition [51]: (a) defines the GraphQL service schema with the available operations (query and mutation); (b) defines the query types (humans, droid), which receive scalar type arguments and return a list of type objects; (c) defines the mutation (createHuman), which receives an input type argument and returns an object type; (d) define the enum type (Episode), which specifies a set of values; (e) defines the type object (Character) that contains a set of fields. The symbol (!) in the appearsIn field

```
extend type Character {
    isHiddenLocally: Boolean
}
```

Fig. 6. Extend type system example.

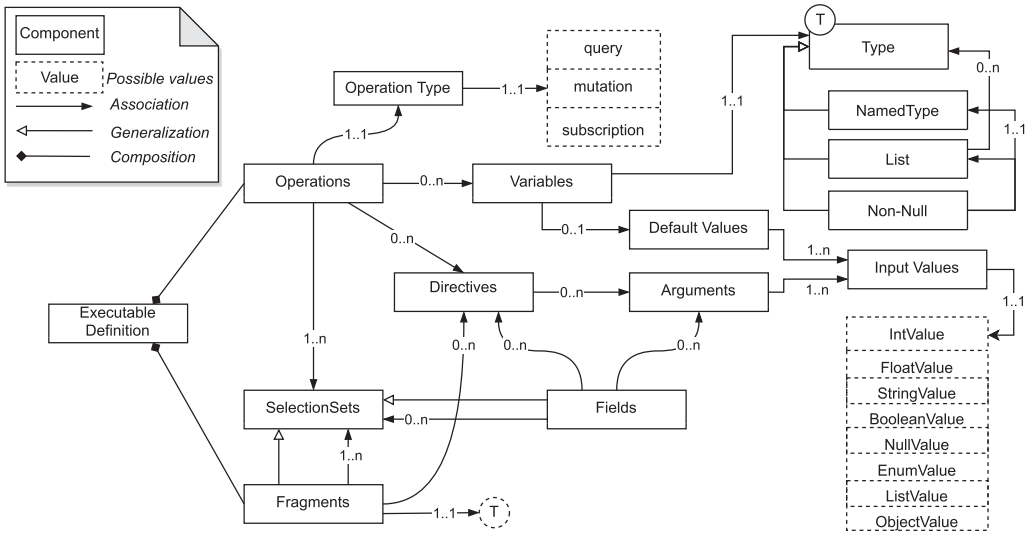


Fig. 7. GraphQL executable definition.

means that it does not accept null values; (f) defines the type Interface (Character) that contains additional fields to the type object Character; (g) defines the implementation (Human) of the interface (Character) that contains the fields of the interface and an additional field (totalCredits); (h) defines the implementation (Droid) of the interface (Character) that contains the fields of the interface and additional field (primaryFunction); (i) defines the input object (HumanInput) that contains a set of fields used in the createHuman mutation; and (j) defines SearchResult as a union type used to search for data in Human or Droid types.

Type System Extension. Represents a GraphQL type system that extends from an original type system. This construct can be used, for example, by a local service to represent data that a GraphQL client only accesses locally, or by a GraphQL service that is itself an extension of another GraphQL service. Figure 6 shows an example of an Object type extension that represents a Character type that has been extended from the original type (see Figure 5), which also adds a local data field.

Executable Definition. Executable documents are defined by operations or fragments [52]. Figure 7 shows the relationship and conceptual structure of their components. In the following, we describe the components of executable definitions.

Operations are the main elements of executable definitions and can be (i) queries, which are read-only and fetch data, (ii) mutations that first write and then fetch data, or (iii) subscriptions, which are long-running requests that fetch data in response to events from the source.
Variables are introduced to maximize the reusability of GraphQL queries parameterized with variables, avoiding costly string building in clients at runtime.
Input Values can be scalar values, enumeration values, lists, or input objects.
Directives are used in a similar way as in the Type System components.

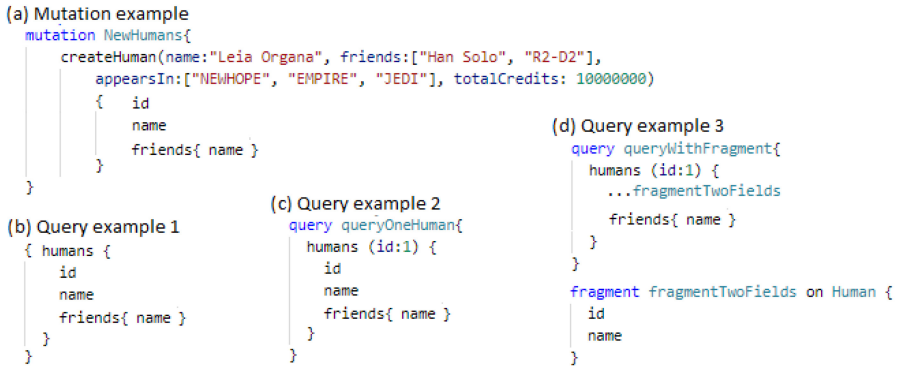


Fig. 8. GraphQL executable definition examples.

Arguments alter the behavior of fields that occasionally accept them. Function arguments are often mapped within a GraphQL server implementation.

Selection Sets are primarily composed of fields. Their purpose is to restrict requests to select only the needed subset of information, avoiding over-fetching and under-fetching data.

Fields describe the discrete information available and can represent complex data or relationships with other data; they are also considered conceptual functions that return values.

Fragments are the basic unit of composition in GraphQL that allows reusing of commonly repeated selections of fields.

Figure 8 shows examples of the most used operations (queries and mutations) of the executable definition [51] concerning the type system defined in Figure 5, namely: (a) defines the operation (NewHumans) that executes the mutation (createHuman). It also shows three ways to execute the type of query (humans); (b) executes the query but without defining the name of the request; (c) defines the request (queryOneHuman) that executes the query "humans" that receives parameter "id" with the value of one, which means that the query will return the record that has that identifier; and (d) defines the request (queryWithFragment) that executes the query using the fragment (FragmentTwoFields) that refers to a set of specific fields of the implementation (Human). Client tools are responsible for sending these definitions to the GraphQL service, which will execute the requests.

2.2.2 Responses. A GraphQL operation response is a map commonly in JSON format, containing three entries: data, errors, and/or extensions. Data contains the result of the execution of the requested operation. Errors appear when the execution has found an error. The extension is reserved for implementers to extend the protocol without content restrictions [43, 51, 52]. Figure 9 shows the response to requests for execution of the mutation and query operations carried out in Figure 8.

2.2.3 Introspection. A GraphQL server supports the introspection of its schema by querying the type system using the same GraphQL language [52]. Figure 10 shows (a) the introspection query to the type system (Figure 5) and (b) the query's answer.

2.2.4 Validation. GraphQL verifies if a request is syntactically correct and ensures that it is unambiguous and mistake-free in a given GraphQL schema. The validation is performed before execution. However, a GraphQL service may execute a request without explicitly validating it if that exact same request is known to have been validated before. GraphQL execution will only

```
{
  "data": {
    "humans": [
      {
        "id": 1,
        "name": "Leia Organa",
        "friends": [ "Han Solo", "R2-D2" ]
      }
    ]
  }
}
```

Fig. 9. GraphQL response example.

(a) Query to GraphQL Introspective

```
query queryToIntrospective {
  __type(name: "Character") {
    kind
    name
    fields { name }
  }
}
```

(b) Response - Query to GraphQL Introspective

```
{ "data": {
    "__type": {
      "kind": "OBJECT",
      "name": "Character",
      "fields": [ "name", "Episode" ]
    }
}
```

Fig. 10. GraphQL introspection example.

consider the executable definitions of Operations and Fragment. Type system definitions and extensions are not executable and therefore are not considered during execution [52].

2.2.5 Execution. GraphQL generates a response to a request via its execution facility in the context of the universe of data available in a schema provided by the appropriate GraphQL service. Only requests that pass validation rules are actually executed; if it has validation errors, then they will be reported in the list of "errors" within the response, and the request will fail without execution [52]. It is worth mentioning that there are cases of requests that pass the validation but during the execution present data errors; in those cases, the execution returns the data result but also returns the description of the error occurred.

Each field of each type is backed by a function called the ***resolver***, which the GraphQL server developer has to provide. When a field is executed, the corresponding resolver is called to produce the response value [43, 52].

3 RESEARCH METHOD

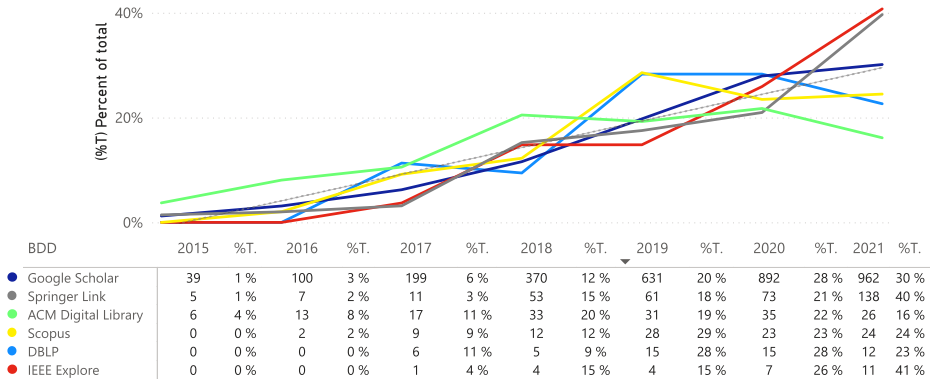
In software engineering, two techniques are clearly distinguished for carrying out studies of the literature that serve to structure knowledge systematically and reproducible, namely, systematic mapping studies and systematic literature review [26].

On the one hand, a SMS is a method for constructing a classification scheme for topics studied in a field of interest. By counting the number of publications for categories within a scheme, the coverage and maturity of the research field can be determined. The results are presented on graphical maps showing the number of publications in different categories. Mapping studies generally cover a more extensive range of publications as the analysis focuses on summaries and key terms [40].

On the other hand, a ***Systematic literature review* (SLR)** is a means of identifying, analyzing, and interpreting reported evidence related to a set of specific research questions in an unbiased and (to some extent) repeatable manner. Unlike mapping studies, systematic reviews generally cover a smaller and more specific range of publications, while the analysis focuses on the details of published contributions [25].

3.1 Need for a Systematic Mapping Study

We approached the need for this study from the point of view of the scientific and industrial community. On the one hand, concerning the scientific community, we searched for SLRs or SMSs



BDD	2015	%T.	2016	%T.	2017	%T.	2018	%T.	2019	%T.	2020	%T.	2021	%T.
Google Scholar	39	1 %	100	3 %	199	6 %	370	12 %	631	20 %	892	28 %	962	30 %
Springer Link	5	1 %	7	2 %	11	3 %	53	15 %	61	18 %	73	21 %	138	40 %
ACM Digital Library	6	4 %	13	8 %	17	11 %	33	20 %	31	19 %	35	22 %	26	16 %
Scopus	0	0 %	2	2 %	9	9 %	12	12 %	28	29 %	23	23 %	24	24 %
DBLP	0	0 %	0	0 %	6	11 %	5	9 %	15	28 %	15	28 %	12	23 %
IEEE Explore	0	0 %	0	0 %	1	4 %	4	15 %	4	15 %	7	26 %	11	41 %

Fig. 11. Publication trend by year.

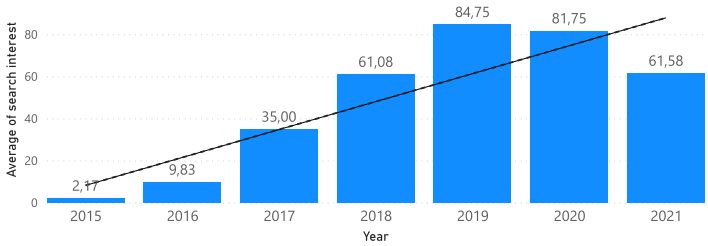


Fig. 12. GraphQL's search trend.

studies on GraphQL in the most used scientific literature databases related to software engineering: ACM Digital Library,[6] SpringerLink,[7] IEEE Xplore Library,[8] Scopus,[9] Google Scholar,[10] and DBLP[11] [9, 25, 26], where we verified that there is no study of this type in the scientific literature. Then, we verified the GraphQL study trend by conducting a survey on the frequency of research in bibliographic databases from 2015 (GraphQL release to the community) to 2021 (last full year). Wherein, Figure 11 shows the trend of publications by year, tabulating the number of publications related to GraphQL, which are normalized in the figure by using the yearly percentage distribution of total publications about GraphQl during the 2015–2021 period in a given database (%T.). We evidenced an average growth of 66% by calculating the percentage variation [31] by year.

On the other hand, regarding the industrial community, Figure 12 sums up the trend of search interest for the term "graphql" using the Google Trends tool,[12] where the search interest is scaled on a range from 0 to 100 based on the proportion of searches for the topic [20]. Although the last two years show a slight decrease, the overall average of the yearly percentage variation calculation [31] increases 98%.

[6]See https://dl.acm.org.
[7]See https://link.springer.com.
[8]See https://ieeexplore.ieee.org.
[9]See https://www.scopus.com/.
[10]See https://scholar.google.com.
[11]See https://dblp.org.
[12]See https://trends.google.com.

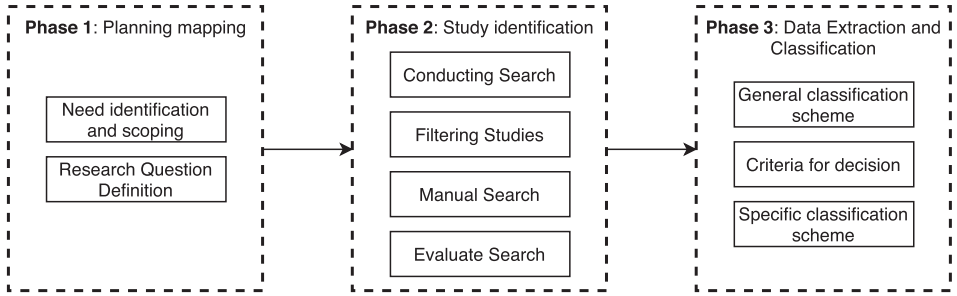


Fig. 13. SMS process definition.

In summary, we found that the scientific and industrial communities have shown increasing interest in GraphQL, but to the best of our knowledge, there are still no literature reviews to provide an overview of the study of GraphQL. As a result, we found the need to conduct an SMS to establish an inventory of classified primary studies (i.e., studies that obtain empirical evidence on a topic of interest [19]) to provide an overview that allows researchers to discover gaps and trends about the study of GraphQL.

3.2 SMS Process Definition

The process for conducting SMS (summarized in Figure 13) was established based on the guidelines proposed by Petersen in 2008 [40], Petersen 2015 [42], and Kuhrmann 2017 [26], which consists of the following phases:

3.3 Phase 1: Planning the Mapping

This section plans the process for conducting the mapping and will serve as a guide for executing or reproducing this work:

Identification of need and scope: The systematic mapping study aims to provide an overview of the scientific community's production and find knowledge gaps around the GraphQL paradigm. The SMS will perform a topic-specific classification about GraphQL and a topic-independent classification to identify the trend, evolution, empirical evidence maturity, and types of research publications based on the facets of Petersen et al. [1, 40, 42].

To define the research questions that guide the SMS, we use Kipling's 5W+1H model [24] to abbreviate the questions: Who, Why, What, Where, When, and How. This model is used to know the most important aspects of a story (commonly used in journalism) [14, 24]. Applying the model, we formulated the following **research questions (RQ)**:

RQ1: Who are the authors and institutions researching about GraphQL? The purpose is to identify which authors and institutions are researching this topic to encourage collaboration.

RQ2: Where were the research papers published? We will define publication venues to identify the acceptance and impact of research on the GraphQL paradigm.

RQ3: What is the empirical evidence maturity of the publications on the GraphQL? We will establish what research methods authors use to gather empirical evidence and what types of research were used to verify the maturity of the research topic.

RQ4: When were the research papers published? This question attempts to show the trend and evolution over time of research on the GraphQL paradigm.

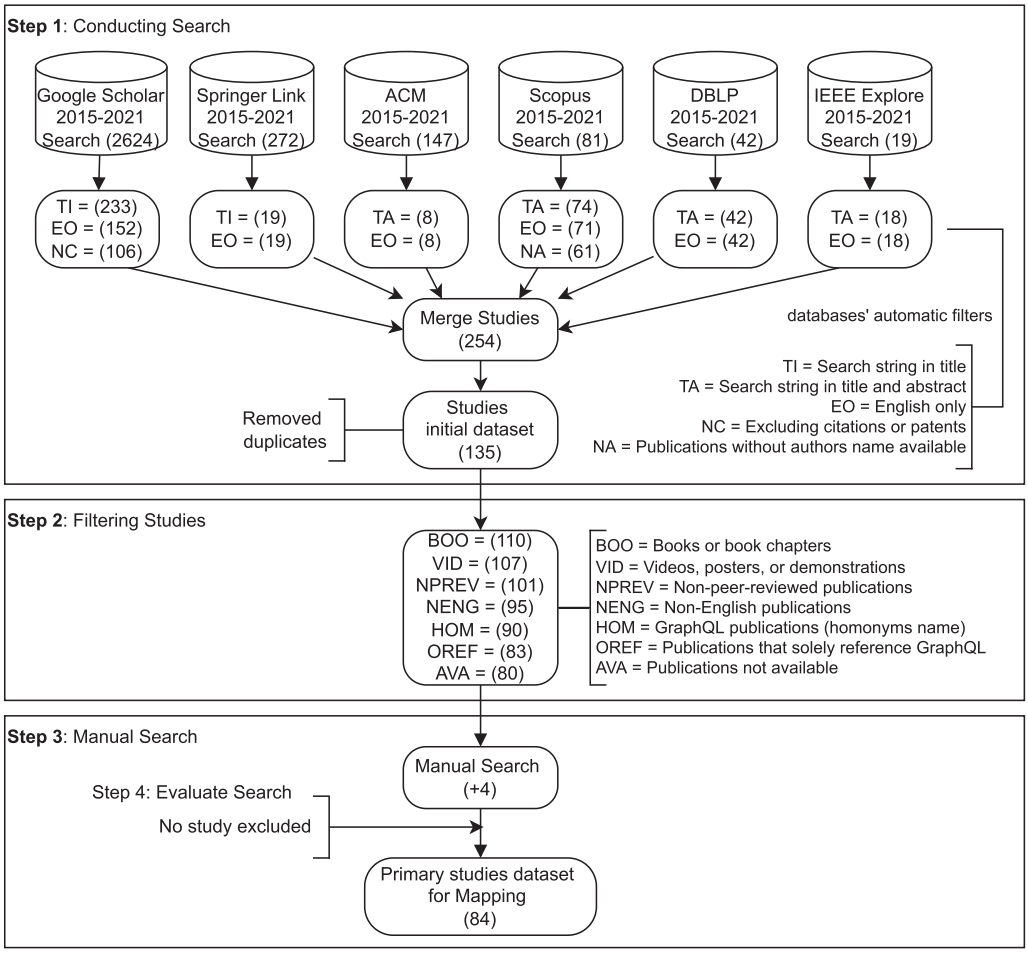


Fig. 14. Process followed to identify the primary study dataset.

RQ5: Why was the GraphQL paradigm studied? This question aims to discover why and in what contexts the scientific community studied the GraphQL paradigm.

RQ6: How was GraphQL introduced to the scientific community? We will classify the type of contributions and the domain of their implementation. We will also identify the components of GraphQL that are most and least used in the research, thus establishing a vision for future research.

RQ7: What are the frontiers of the research on GraphQL? This question abstracts the main findings presented in the studies and the open questions posed as future work.

3.4 Phase 2: Study Identification

This section details how we identified the primary study dataset for mapping; see Figure 14.

Search Strategies: We chose two strategies, the first being an automatic search in the databases and then supplemented with manual searching to add relevant studies that did not appear in the first search. The steps we established for the search are: (1) Conduct the search for primary studies using a search string in the databases chosen for the study; then apply the first study filter using

the databases' automatic filters. After that, using bibliographic reference management tools (e.g., Zotero and Mendeley), we merge the studies into a homogeneous structure. We finalized this step by removing duplicate studies. (2) Filter the studies by applying inclusion and exclusion criteria. (3) To improve the quality of the primary study dataset, we initially piloted the snowballing search strategy [58] but found no new publications that met the inclusion criteria of the study filtering. Therefore, we followed the recommendations of Wohlin [58] to use the manual search strategy to add relevant studies to the dataset. (4) Finally, evaluate the search.

Conducting search: To establish the search string, we use the interactive improvement approach by identifying keywords from known papers. We decided to establish the search string with the word "graphql" to cover everything reported in the scientific community since 2015, when GraphQL was released to the community, until June 30, 2021. We identified that the appropriate scientific population to consult for primary studies is the scientific literature databases commonly used in software engineering: ACM Digital Library, Springer Link, IEEE Xplore Library, Scopus, DBLP, and Google Scholar [9, 25, 26, 42]. We began conducting the automatic search in the databases and obtained a total of 3,185 studies. After that, we searched again and applied the databases' automatic filters (see TI, TA, EO, NC, and NA in Figure 14), obtaining 254 studies. Then, we removed duplicate studies obtaining an initial dataset of 135 studies.

Filtering Studies: In this step, we define and apply the inclusion and exclusion criteria on the study's initial dataset to establish the relevance of the research topic [27, 42].

Inclusion criteria:

- Publications in English that have been peer reviewed in journals, conferences, or workshops.
- Gray Literature (evidence not controlled by commercial publishers) can include academic papers, including theses and dissertations [38], such as Bachelor's, Master's, and Ph.D. theses.
- Publications from 2015 to 2021.
- Publications that focus on the use of the GraphQL paradigm in the title and the abstract.
- Publications that contain a reasonable context, objectives, and research method.

Exclusion criteria:

- **Books or book chapters (BOO)**.
- **Videos, posters, or demonstrations (VID)**.
- **Non-peer-reviewed publications (NPREV)**.
- **Non-English publications (NENG)**.
- Publications that refer to GraphQL proposed by Huahai He and Ambuj Singh in 2008 (**homonyms name (HOM)**). It is a graph query language that allows flexible manipulation of graph structures. In addition, they introduce the notion of formal languages for graphs (useful for the definition of graph queries as well as database graphs) [22].
- Publications that solely **reference GraphQL as an external citation (OREF)** and they do not use or extend the paradigm (Section 2.1).
- Publications **not available (AVA)** through our institutions' subscriptions or in open repositories.

After applying the inclusion and exclusion criteria, we obtained a dataset of *80* studies

Manual search: We complemented the search strategy with the manual search method; we found four studies in the results of the automatic searches before applying the automatic filters in the databases; this resulted in a dataset of 84 primary studies.

Evaluation of the search: Petersen [42] and Kitchenham [25] recommend maintaining a low stringency in evaluating the quality of primary studies, because SMSs aim to provide an overview

Table 1. GGS Conference Rating

Class	Ratings	Description
1	A++, A+	top notch conferences
2	A, A−	very high-quality events
3	B, B−	events of good quality
—	Work in progress (WIP)	work in progress

of the research topic area. For this reason, we decided not to exclude any primary studies in this section, resulting in a dataset of *84* primary studies for mapping.

3.5 Phase 3: Data Extraction and Classification

In this phase, we extracted data from the primary studies to conduct a general classification that does not depend on the researched topic (topic-independent classification) and another specific classification about the GraphQL paradigm (topic-specific classification) [42].

Topic-independent classification scheme: The following describes the fields of data extraction based on the *facets*, *venue*, *research type*, *research method*, and *study focus* proposed in Petersen et al. [42]:

Title: title of the publication, specifying the study, use, or extension of the GraphQL paradigm.
Abstract: summary of the publication, specifying the study, use, or extension of the GraphQL paradigm; do not use the term GraphQL only to reference an external citation.
Authors: principal author (first author) and co-authors of the publication.
Authors' affiliation: first affiliation of the publication's authors.
Countries: country of authors' affiliation.
Year: year of the publication.
Venue: name of publication venue.
Venue types: peer-reviewed venues such as journals, conferences, and workshops [42]. In addition, we added gray literature such as academic articles, theses, and dissertations to reduce possible publication bias and provide a more balanced view of the evidence [38].
Venue rate: impact factor of the venue types measured with **Scimago Journal & Country Rank (SJR)**,[13] **Journal Citation Reports (JCR)**,[14] and **GII-GRIN-SCIE (GGS)** Conference Rating.[15] SJR calculates the Scimago Journal Rank impact factor of journals and countries based on information from the Scopus database (Elsevier B.V.) [48], measures the weighted citations of a journal according to the scientific area and the relevance of the citing journals [49]. Journal subdisciplines are ranked into four quartiles (Q1 to Q4) using the SJR Citation Index [11]. JCR is an ISI Web of Knowledge product that provides a rich array of citation metrics such as the **Journal Impact Factor (JIF)** [5]. A journal's quartile ranking (Q1 to Q4) is determined by comparing a journal to others in its JCR category based on JIF [6]. GGS is an initiative that provides a unified rating of computer science conferences [37]; see Table 1.
Publication types: for peer-reviewed types of venues are journal articles, conference papers, and workshop papers [42]. For grey literature, according to the Finnish Ministry of Education thesis classification:[16] Bachelor's, Master's, and Ph.D. theses

[13] https://www.scimagojr.com/.
[14] https://jcr.clarivate.com/.
[15] http://gii-grin-scie-rating.scie.es/.
[16] https://aaltodoc2.org.aalto.fi/doc_public/ohjeet/publicationclassification2010.pdf.

Table 2. Research-type Classification

Research type \ Conditions	*Used in practice*	*Novel solution*	*Empirical evaluation*	*Conceptual framework*	*Opinion about something*	*Authors' experience*
Evaluation research	T	•	T	•	F	•
Validation research	F	•	T	•	F	•
Solution proposal	•	T	F	•	F	•
Philosophical papers	F	F	F	T	F	F
Experience papers	T	F	F	•	F	T
Opinion papers	F	F	F	F	T	F

Legend applies condition: T = True, F = False, and • = irrelevant or not applicable.

Research type: We base it on the classification of research types proposed by Wieringa [57] and complement it with the decision table to disambiguate the classification of studies by checking a series of conditions proposed by Petersen et al. [42]; see Table 2. Next, we show and order the research types according to the maturity exposed by the studies [57, 59]:

Evaluation research, empirically evaluates the investigation of a problem or application of a technique in engineering practice. *Validation research*, develops a novel solution and is empirically evaluated in a laboratory setting (i.e., not used in practice). *Solution proposal*, this study proposes a solution to a research problem, and the benefits are discussed but not evaluated. *Philosophical papers*, structures an area in the form of taxonomy or conceptual framework, hence provides a new way of looking at existing things. *Experience papers*, includes the author's experience of what and how something happened in practice. *Opinion papers*, these studies contain the author's opinion on a particular issue without relying on related research papers and methodologies.

Research methods: Petersen et al. [42] identify that the types of "evaluation research" and "validation research" need empirical evaluation by using different research methods frequently applied in software engineering [10, 21, 57, 60]; in this sense, we analyze and use the following subset of research methods proposed by Reference [42]: *Survey,* identify the characteristics of a broad population of individuals. It is most closely associated with the use of questionnaires for data collection [10]. *Case study,* investigate a single entity or phenomenon in its real-life context within a specific time-space [60]. *Controlled experiment,* investigates a testable hypothesis where one or more independent variables are manipulated to measure their effect on one or more dependent variables [10]. *Simulation,* is a powerful technique to handle the complexity of the model (hundreds of dynamic variables and causal linkages). The use of simulation enables managers to assess quickly and safely the implications of an intended policy before it is implemented [30]. *Prototyping,* is an approach based on an evolutionary view of software development and an impact on the development process. It involves producing early working versions (prototypes) of the feature application system and experimenting with them [4]. *Mathematical analysis,* covers the basic techniques to identify a set of reasoning rules in a precise (and therefore mathematical) way in the system under study [3].

Study setting: This is the context in which the study was conducted, whether in academia, industry, or government [7]. In this sense, we established the following semantics "Academic": studies performed in synthetic environments that do not implement solutions

for public use; "Industry" and "Government": studies conducted in industry/government practice environments or they expose solutions for public service.

Finding: The study's findings are usually in the results or conclusions study sections.

Challenge: Challenges posed by the authors are usually in the discussion, conclusions, or future work sections of the study.

Topic-specific classification scheme: The following describes the data extraction fields for the specific classification of the GraphQL paradigm:

Knowledge area: Software engineering knowledge areas based on the Guide to the Software Engineering Body of Knowledge SWEBOK version 3.0 of the IEEE Computer Society [3] are: Requirements, Design, Construction, Testing, Maintenance, Configuration Management, Engineering Management, Engineering Process, Engineering Models and Methods, Quality, Engineering Professional Practice, Engineering Economics, Computing Foundations, Mathematical Foundations, and Engineering Foundations.

Contribution domain: This is the domain of the GraphQL service used in the study's technical contribution. We identify the Provider and Client domains. The Provider domain is any contribution implemented in the "GraphQL service" see Figure 2, and the Client domain is the contributions consuming the GraphQL service, see "Client tools" in Figure 2.

Contribution type: This is a high-level classification of GraphQL technical contributions. We define the following semantics to identify and classify contribution types: *Implementation,* refers to studies that use the GraphQL paradigm to implement software. *Analysis,* refers to studies that contribute a comparative analysis or theoretical review to identify ways to apply the GraphQL paradigm. *Integration,* refers to studies that present the integration of GraphQL with other technologies to design or build integrated solutions. *Extension,* refers to studies that propose a new component or functionality in the GraphQL paradigm. *Applications,* refer to studies that propose solution approaches applying the GraphQL paradigm but without its implementation.

GraphQL Components: Based on Section 2, we established the classification by "Service" with its components: Execution, Introspection, Resolvers, Type System, Validation. "Type System Definition" with its components: Directives, Enums, Input Object, Interfaces, Objects, Scalars, Schemas, Unions. "Execution Definition" with its components: Fragments, Mutations, Query, and Subscription.

Public API: This is the actual public API used in the publication, e.g., as a use case or supporting system.

Consequently, we established the structure for data collection in this study based on the research questions presented; for each question, as shown in Table 3, we define a set of fields derived from the topic-independent classification, topic-specific classification, and minimum data structures recommended by Kuhrmann et al. [26] and Petersen et al. [42].

3.6 Validity Evaluation

In conducting the SMS, we tried to be as methodological as possible; therefore, we evaluated the validity based on the guidelines of References [41, 42], using the following considerations: *Descriptive validity* is the description with precision and objectivity. We minimize this threat by introducing decision criteria forms for general categorization and classification based on the Petersen guide [42] (see Section 3.5). We also introduced topic-specific classification criteria based on the GraphQL paradigm components exposed in this article (see Section 2). *Theoretical validity* determine the ability to capture what we intend to capture [42]. To minimize this threat, in the

Table 3. Data Extraction Structure

No.	Field	Cardinality	Description	RQ
1	No.	1	Publication number	
2	Title	1	Publication title	
3	Abstract	1	Summary of the publication	
4	Authors	1...n	Authors of publication	RQ1
5	Authors affiliation	1...n	Authors' institution of affiliation	RQ1
6	Countries	1...n	Country of the authors' institutions	RQ1
7	Venue	1	Publication venue name	RQ2
8	Venue type	1	Classification of venues	RQ2
9	Publication type	1	Types of publication in venues	RQ2
10	Venue acronym	1	Acronym for the publication venue name	RQ2
11	Venue rate	1	Venue rate	RQ2
12	Research type	1	Research type classification	RQ3
13	Research method	1	Classification of research methods	RQ3
14	Year	1	Year of publication	RQ4
15	Study setting	1	Context in which the study is applied	RQ5
16	Knowledge area	1	SWEBOK Knowledge areas	RQ5
17	Contribution domain	1	Domain in which the contribution is made	RQ6
18	Contribution type	1	Contribution type of the publication	RQ6
19	Contribution	1...n	Technical contribution of the publication	RQ6
20	GraphQL Components	1...n	GraphQL components in the publication	RQ6
21	Public API	1...n	Public APIs used in the publication	RQ6
22	Finding	1...n	The study's findings	RQ7
23	Challenge	1...n	Challenges posed by the authors	RQ7

Study Identification, we established the search string only with the word "graphql" to obtain an adequate sample concerning the target population. We also chose two strategies for the search: We started with the database search; we complemented it with the manual search strategy, where we found four studies that we added to the initial study dataset. We tried to minimize the risk of bias in data extraction and classification when the first researcher performed an individual mapping followed by a second researcher's review. Differences found were validated by a joint reading of the full text of the publication and then discussed to a consensus based on the established rules for classification in Section 3.5. *Generalizability*, according to Petersen and Gencel [41], should be considered internal generalization (within a population) and external generalization (between different populations). Internal validity ensures that a researcher's experimental design closely follows the cause-and-effect principle; therefore, in this study, we followed most of the methodological recommendations exposed by Petersen's guide [42]. However, the existence of manual classifications in the process exposes us to introduce some study selection errors. Therefore, we considered a wide range of papers (including the gray literature) and introduced methods of decision criteria in the classifications to minimize this impact. External validity measures the applicability of the study results to other situations, groups, or events (generalizability). We tried to minimize this threat by using the Petersen et al. guide [42], which is very popular for conducting SMSs in software engineering. Besides, we conducted the specific classification of GraphQL components using the GraphQL paradigm based on the official site of the formal GraphQL specification. *Interpretive validity* is when the conclusions are drawn reasonably, given the data. We minimize the threat of bias in interpreting the conclusions by exposing the results described

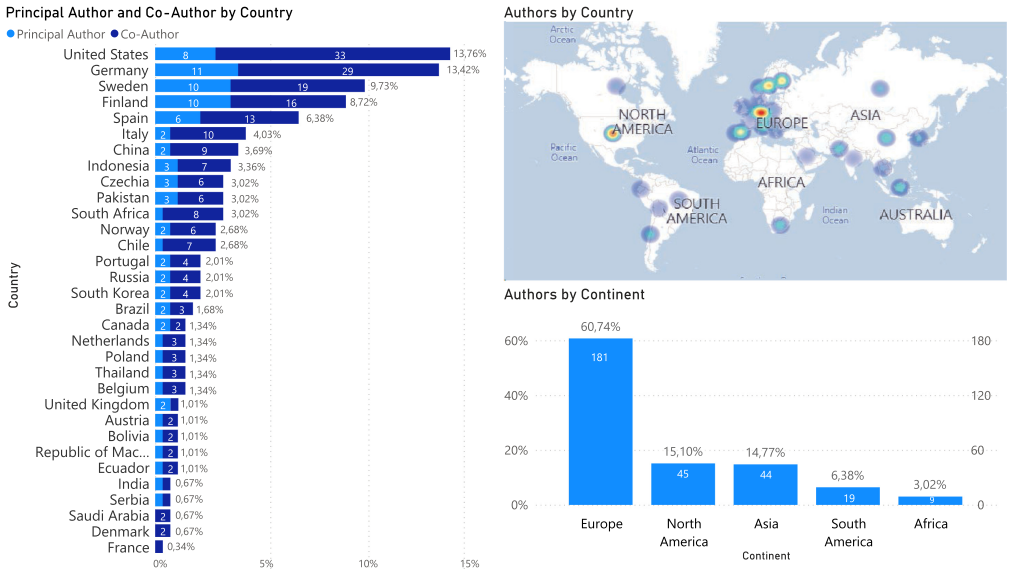


Fig. 15. Authors by continent and country.

by the first author, then reviewed separately by the three co-authors, and then discussed together in consensus meetings to unify revisions and interpretations. *Repeatability* requires detailed information on the research process. In this study, we report in detail the SMS process, expose the work files in digital repositories, and explain the measures taken to reduce possible threats to validity.

4 RESULTS

In this section, we answer the RQ established in Section 3.3 using the results of the data extraction process from the primary studies conducted in Section 3.5. The list of the primary studies is available in the supplementary material of the article [45], where the 84 studies are coded from study *"S01"* to study *"S84."*

4.1 RQ1: Who Are the Authors and Institutions Researching About GraphQL?

In this question, we discovered the most active researchers and institutions that researched the GraphQL paradigm. On the one hand, we show the authors' information who used the GraphQL paradigm in their publications. We considered the publication's first author as the principal author and the other authors as co-authors, obtaining 298 authorships in the publications (84 main authorships and 214 co-authorships) corresponding to 270 researchers. Figure 15 shows a summary of authors by country and continent; we observe that authors affiliated with European institutions lead the number of contributions with 60.74%; however, the United States is the country with the most authors, followed by Germany, which highlights the number of authorships of the principal authors. Table 4 shows the authors with the highest number of publications.

On the other hand, we show the institutions of the authors' affiliation. Figure 16 shows the institutions with two or more publications; the others are grouped in "Others." There are 86 institutions from 33 countries, where the Linkoping University of Sweden contributed the highest number of publications, followed by the Aalto University of Finland, IBM Research of United States, and RWTH Aachen University of Germany.

Table 4. Top Authors by Publications

Author	Publications as principal author	Publications as co-author	Total publications
Hartig Olaf	3	3	6
Wittern Erik	2	1	3
Cha Lan	1	2	3
Taelman Ruben	2	—	2
Brito Gleison	2	—	2

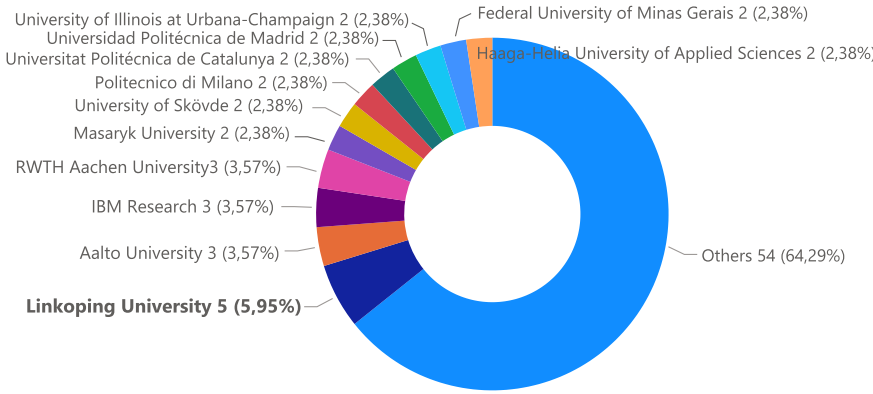


Fig. 16. Publications by affiliation.

4.2 RQ2: Where Were the Research Papers Published?

This question identifies the venues, venue types, publication types where GraphQL studies were published, and their impact. In this sense, we started by identifying and ordering the following venue types according to the rigor of the review of publications; it is worth mentioning that we considered the gray literature to verify the interest of academia in the study of GraphQL. The publication types found at the venue are bachelor's and master's theses, workshop papers, conference papers, and journal papers. However, in conferences and workshops, we identified two ways of publishing their studies; the first is to publish the studies in the proceedings of the venues, and the second way is to publish the proceedings in a journal volume so that the impact of the journal catalogs the impact of their studies. Therefore, we subdivide conference and workshop papers whose proceedings are published in a journal: conference paper (journal) and workshop paper (journal).

Figure 17 shows the publications by publication type and venue type. We observed that most publications (46.07%) are in conferences, and peer-reviewed publications are 69.05%, and the academic review is 30.95%.

The following shows the publications' impact was measured by: GGS, SJR, and JCR. In this section, we clarify that the impact of the gray literature is not measurable; therefore, the analysis of papers from workshops, conferences, and journals. Table 5 shows the workshop papers and their impact measured in GGS and SJR. The **Workshop on (Constraint) Logic Programming (WLP)**, published in the journal Electronic Proceedings in Theoretical Computer Science, is highlighted with an SJR:0.33.

Table 6 shows the conference paper impact measured with GGS and SJR. According to the SJR metrics, on the one hand, the conferences that index their proceedings directly in Scopus have an SJR index but no quartile; in this segment, the leading conference is SANER (SJR: 0.29). On

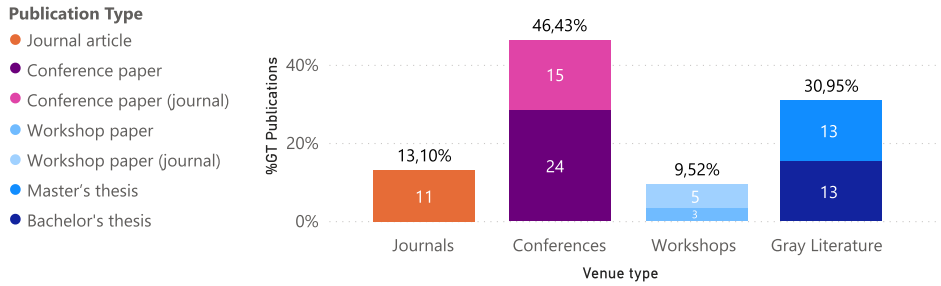


Fig. 17. Publications by venue type and publication type.

Table 5. Workshop Papers and Their Impact

Publication Type	GGS Class	SJR Quartile	SJR Year	SJR	Publication Number	Publications
Workshop paper	—	—	—	—	3	S06, S08, S39
Workshop paper (journal)	WIP	—	2019	0.18	3	S01, S02, S26
	—	—	2020	0.18	1	S62
	—	—	2019	0.33	1	S17

Table 6. Conference Papers and Their Impact

Publication Type	GGS Class	SJR Quartile	SJR Year	SJR	Publication Number	Publications
Conference paper	Class 1	—	—	—	3	S28, S32, S47
	Class 3	—	—	—	3	S25, S40, S57
	WIP	—	—	—	6	S11, S15, S38, S63, S76, S78
	—	—	2020	0.29	1	S24
	—	—	—	—	11	S33, S35, S41, S55, S56, S64, S67, S68, S71, S81, S83
Conference paper (journal)	Class 2	Q2	2019	0.43	3	S07, S12, S30
	Class 3	Q2	2019	0.43	2	S13, S21
	WIP	Q3	2020	0.25	1	S73
	WIP	Q3	2019	0.18	1	S69
	—	Q2	2019	0.43	1	S48
	—	Q3	2019	0.19	2	S19, S37
	—	Q4	2020	0.16	1	S61
	—	—	2019	0.20	2	S09, S18
	—	—	2019	0.34	1	S31
	—	—	—	—	1	S66

the other hand, conferences that publish their proceedings in volumes of journals indexed in Scopus generally have SJR and quartile indexes. In this segment, the leading conferences are CAISE, ICSOC, ICWE, and MEDI, published in the journal "Lecture Notes in Computer Science," which has quartile 2 (Q2) and SJR: 0.43. According to the GGS metrics, 19 of the 39 conference papers (48.72%) were ranked. In this segment, the top conferences at level 1 are WWW (Class 1: A++) and ESEC/FSE (Class 1: A+).

Table 7. Journal Papers and Their Impact

Journal Name	ISSN	JCR-SJR Year	Quartile -JIF (JCR)	Quartile -SJR	Publications
Journal of Molecular Biology	0022-2836	2020	Q1-5.47	Q1-3.19	S79
Journal of Cheminformatics	1758-2946	2019	Q1-5.32	Q1-1.43	S05
IT Professional	1520-9202	2019	Q1-3.70	Q1-0.63	S14
Molecules	1420-3049	2020	Q1-4.41	Q1-0.78	S84
PLoS ONE	1932-6203	2020	Q2-3.24	Q1-0.99	S16
Electronics	2079-9292	2019	Q2-2.41	Q2-0.30	S58
BMC Medical Informatics and Decision Making	1472-6947	2019	Q3-2.31	Q1-0.91	S20
Software and Systems Modeling	1619-1366	2020	Q3-1.91	Q2-0.42	S60
Software Engineering and Knowledge Engineering	0218-1940	2019	Q4-0.89	Q3-0.25	S42
Journal of Object Technology	1660-1769	2019	—	Q3-0.24	S46
Theoretical And Applied Science	2308-4944	—	—	—	S75

Table 8. Publications by Research Type and Research Method

Research type	Research method	Publications
Evaluation research	Case Study	S03, S05, S12, S18, S34, S54, S55, S70, S74, S83
	Experiments	S63, S78, S81
	Prototyping	S20
	Survey	S32
Validation research	Experiments	S09, S13, S14, S28, S29, S37, S38, S41, S42, S43, S44, S47, S56, S58, S62, S65, S77, S80
	Prototyping	S07, S11, S27, S31, S35, S46, S48, S59
	Survey	S01, S10, S24, S30, S40
	Case Study	S16, S33, S39, S60
	Mathematical Analysis	S02, S08
Solution proposal	—	S21, S25, S45, S57, S69, S72, S73
Experience paper	—	S15, S52, S64, S67, S76, S79, S82, S84
Opinion paper	—	S19, S22, S23, S26, S36, S49, S50, S51, S53, S61, S66, S68, S71, S75

Table 7 shows the journal articles' impact measured with JIF of JCR and SJR metrics. The highest quartile 1 (Q1) journals are Journal of Molecular Biology (JIF: 5.47, SJR: 3.19), Journal of Cheminformatics (JIF: 5.32, SJR: 1.43), and IT Professional (JIF: 3.70, SJR: 0.63).

4.3 RQ3: What Is the Empirical Evidence Maturity of the Publications on the GraphQL?

This question identifies the maturity of the empirical evidence of the studies by classifying the publications' rigor; therefore, we classified the publications by the research type according to the criteria indicated in Table 2 and identified the research methods (see Section 3.5) used to support their GraphQL paradigm findings. Table 8 shows the publications by research type and methods.

Figure 18 shows, on the one hand, that most of the publications have a strong empirical focus, since they are classified as "evaluation" or "validation" research (61.91%). On the other hand, the

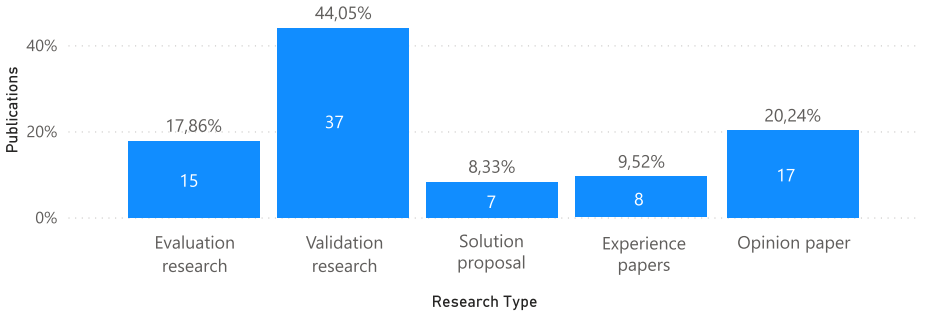


Fig. 18. Publications by research type.

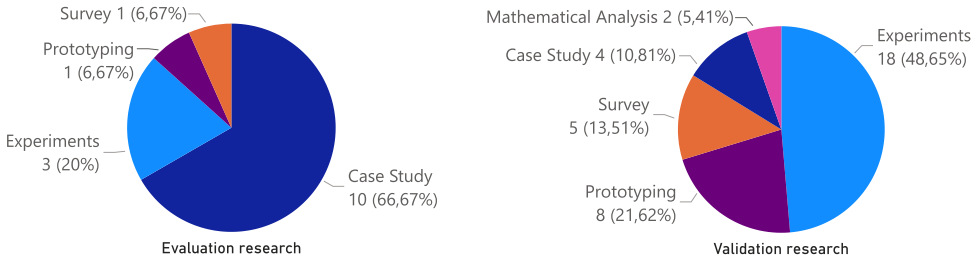


Fig. 19. Publications by research methods in validation and evaluation research types.

remaining publications showed studies without empirical evaluation (i.e., they did not apply the research methods described in Section 3.5), classified as solution proposals, experience papers, and opinion papers. It is worth mentioning that there were no publications of the philosophical paper.

Figure 19 shows the research methods used in "evaluation" or "validation" research; where it is noticeable that the case study is the most commonly used research method in software engineering practice (i.e., evaluation research). In contrast, the most commonly used methods were experiments and prototypes in validation research.

4.4 RQ4: When Were the Research Papers Published?

This question shows the evolution of GraphQL publications from 2015 to June 30, 2021. We consider the studies since 2015, because that is the year GraphQL was launched to the community.

Figure 20 shows the number of studies by type of research and year of publication. Again, we observe that the growth of publications is constant over time, with 2019 being the year with the highest growth (17.86%) compared to 2018. Note that publications of research types that present empirical evidence also increased over the years ("validation" and "evaluation" research types). Thus, although GraphQL is a young research topic and maintains a growing trend, the scientific community should generate more studies with empirical evidence to improve and mature the knowledge base about GraphQL.

4.5 RQ5: Why Was the GraphQL Paradigm Studied?

With this question, we try to determine why researchers conduct studies on GraphQL; in this sense, we assume Petersen's recommendations [42] to answer the question with the following metrics: study setting, application domain, and software engineering knowledge areas based on the SWEBOK.

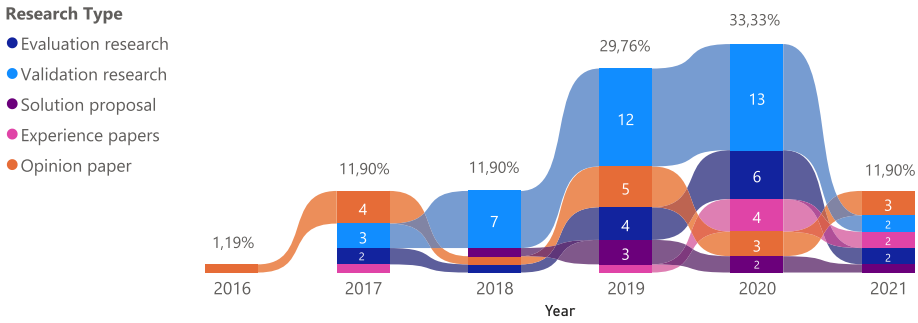


Fig. 20. Publications by research type and year.

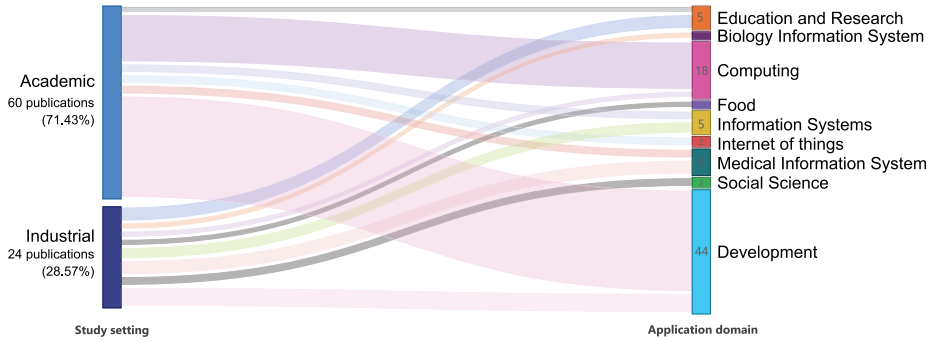


Fig. 21. Publications by study setting and application domain.

Figure 21 shows the publications by study environment and application domain. Most publications (71.43%) were in the academic context and the rest in the industrial context; we did not find applied studies in the governmental context. We also calculated that the predominant application domains were development (52.38%) and computing (21.43%); this indicates that the academic world began to study the technical components of the GraphQL paradigm before applying them to other disciplines or sciences.

Figure 22 and Table 9 show the classification of publications by **software engineering (SE)** knowledge areas based on SWEBOK (see Section 3.5) used in their studies. Note that the knowledge area "construction" is the most applied (55.95%), followed by far by "computing foundations." These results of the knowledge areas match and are complemented by the results of the application domains, ratifying the interest of researchers in the study and technical application of the GraphQL paradigm.

4.6 RQ6: How Was GraphQL Paradigm Introduced to the Scientific Community?

We answer this question by identifying specific classifications of the use and application of the GraphQL paradigm based on Sections 2 and 3.5. In the data extraction and classification phase, we identify the following categories: contribution domain, contribution type, GraphQL components, and public APIs used in the studies. One of the GraphQL expert authors conducted the classification of studies twice to minimize human errors.

Contribution domain: Establishes the application domain of the technical contribution of the GraphQL service in publications; based on Section 2.1. During the classification of the studies,

Table 9. Publications by SE Knowledge Areas

Swebok areas	Publications
Construction	S03, S05, S12, S13, S14, S15, S20, S22, S24, S25, S27, S34, S35, S36, S41, S43, S44, S45, S49, S51, S52, S53, S54, S55, S56, S57, S58, S59, S61, S62, S64, S65, S66, S67, S68, S69, S70, S71, S72, S73, S77, S78, S80, S81, S82, S83, S84
Computing foundations	S01, S06, S17, S26, S28, S29, S30, S31, S32, S37, S40, S42, S75
Design	S04, S07, S09, S11, S18, S19, S21, S23, S38
Models and methods	S16, S46, S47, S48, S60, S76, S79
Testing	S10, S39, S50, S63
Mathematical foundations	S02, S08, S33
Process	S74

Table 10. Publications by Contribution Domains

Contribution domain	Publications
Provider	S01, S02, S03, S07, S08, S09, S10, S12, S13, S14, S15, S17, S20, S21, S23, S25, S28, S29, S32, S33, S34, S37, S38, S39, S40, S41, S42, S43, S44, S46, S48, S49, S50, S53, S55, S59, S47, S60, S61, S62, S63, S65, S66, S67, S68, S69, S70, S71, S72, S75, S78, S79, S80, S81, **S04, S05, S11, S18, S19, S22, S24, S27, S30, S31, S36, S45, S51, S52, S54, S57, S58, S16, S64, S73, S74, S76, S77, S82, S83, S84**
Client	**S04, S05, S11, S16, S18, S19, S22, S24, S27, S30, S31, S36, S45, S51, S52, S54, S57, S58, S64, S73, S74, S76, S77, S82, S83, S84**, S06, S26, S35, S56

Note: Publications marked in bold contribute to both provider and client domains.

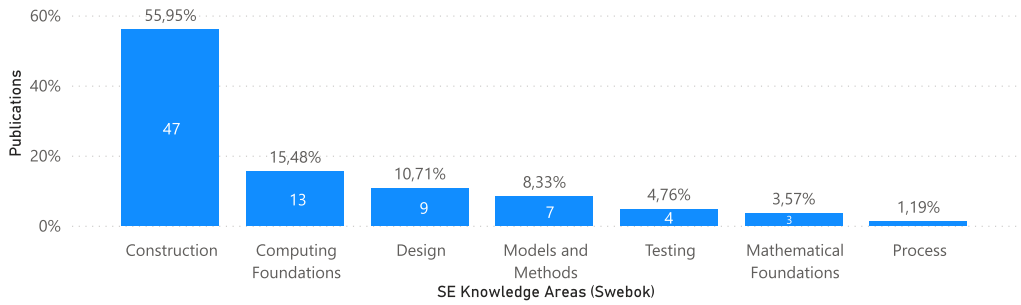


Fig. 22. Publications by SE Knowledge areas.

we identified the *provider* and *client* domains, where 54 publications make contributions in the provider domain, 26 publications in both domains, and 4 publications in the client domain. Table 10 shows the classification of publications by contribution domain. Figure 23 shows (a) the number and percentages of contributions of publications in the domains and their combination; (b) the percentages of contributions grouped by domain.

Contribution type: This is a high-level classification of GraphQL technical contributions in publications. Similar to the previous segment, we found publications that provided multiple contribution types; thus, we classified 127 technical contributions from 84 publications. Table 11 shows

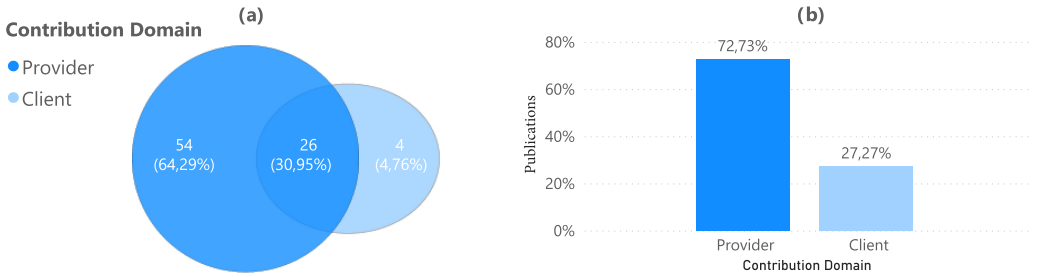


Fig. 23. Publications by contribution domains.

Table 11. Contribution Types in Publications

Contribution type	Contribution	Publications
Implementation	GraphQL Implementation	S05, S15, S23, S35, S36, S38, S51, S52, S54, S57, S60, S64, S68, S69, S70, S72, S79, S81, S82, S83, S84, **S03, S07, S09, S16, S17, S20, S27, S39, S44, S45, S49, S50, S55, S62, S63, S65, S66, S67, S76, S77, S78, S80**
	Implement a Tool	**S10, S32, S62**
	Migration to GraphQL	**S12, S18, S22, S24, S74**
	Wrapper	**S13, S24, S34, S59, S71**
Analysis	Comparison	S01, S04, S26, S29, S30, S37, S40, S43, S56, S75, **S03, S09, S12, S17, S18, S22, S24, S31, S34, S44, S45, S46, S49, S50, S55, S58, S65, S66, S67, S71, S74, S76, S77, S78, S80**
	Analysis	**S10, S47**
	Evaluation	S14, S41, **S13, S32, S59**
	Theoretical Review	S06, S53, S61, S62, **S80**
Applications	Architectural Design	S11, S19, **S58**
	Generate GraphQL	S21, S42, S48, **S25, S31, S63**
	Proposal for using GraphQL	S73, **S39, S46**
	Proposal using GraphQL	**S20, S27**
	Transforming GraphQL	S08, **S59**
Extension	Design Framework	**S07**
	Profile for model	**S16**
	Theoretical Framework	S02, S28, S33, **S47**
Integration	Generate GraphQL	**S25, S31**

Note: Publications marked in bold provide two or more contribution types.

the classification of publications by contribution type and technical contribution, and Figure 24 shows that the most frequent contribution types were implementation and analysis.

Contributions: These are the specific technical contributions using the GraphQL paradigm in the publications. Figure 24 shows that most of the contributions are of the "Implementation" and "Analysis" types; therefore, we will now analyze these two contributions.

On the one hand, type "Implementation" contributions present solutions such as tool implementations, wrappers, migrations, and especially (43 of 56 contributions) GraphQL API

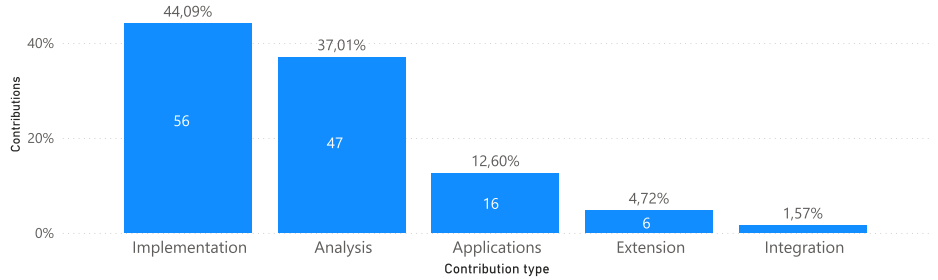


Fig. 24. Contribution types in publications.

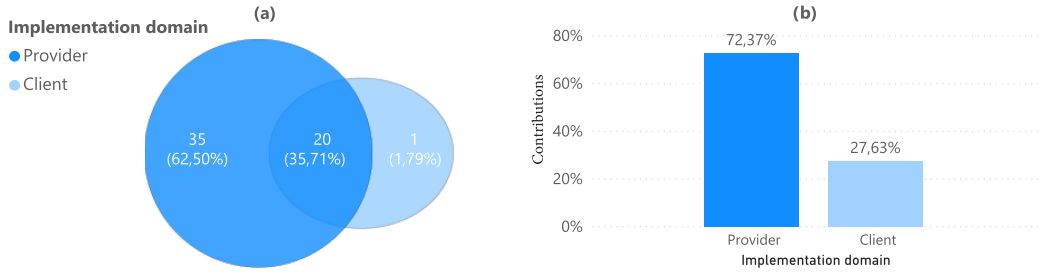


Fig. 25. Type "Implementation" contributions in publications.

Table 12. GraphQL Comparisons

Comparisons	Publications
Between REST and GraphQL	S03, S04, S09, S12, S17, S18, S22, S24, S29, S31, S34,S37, S43, S45, S49, S50, S55, S56, S58, S66, S67, S74, S75, S76, S77, S78, S80
GraphQL APIs schemes	S01, S30
Other comparisons	S26, S40, S44, S46, S65, S71

implementations; where 35 publications perform implementations in the provider domain, 20 publications in both domains, and 1 publication in the client domain. Figure 25 shows (a) the number and percentages of implementations in the domains and their combination; (b) the percentages of implementations grouped by domain.

On the other hand, the contributions of the "Analysis" type presented works such as theoretical reviews, analyses, and the most notable (35 of 47 contributions) comparisons of GraphQL with other technologies. Table 12 shows the different comparisons with GraphQL in the studies, where most of the comparisons are between GraphQL and REST. In this sense, we note that the main reported findings mention that GraphQL is more efficient than REST, because it reduced the time and size of responses in the implemented APIs.

GraphQL Components: In this section, we map the GraphQL components mentioned, used, and exemplified in the publications to understand how the scientific community studied the GraphQL paradigm. The semantics of the classifications we established for the components are: (i) *components mentioned*, which refers to components mentioned in the publications but not necessarily used; and (ii) *components exemplified*, which refers to components that were exemplified but not necessarily used in the publications; (iii) *components used*, which identifies components used as part of the GraphQL technical contribution in the publications. Figure 26

Classification	Component	Mentioned	%	Exemplified	%	Used	%
Service	Execution	42	50 %	26	31 %	40	48 %
	Introspection	28	33 %	8	10 %	19	23 %
	Resolvers	39	46 %	20	24 %	37	44 %
	Type System	36	43 %	12	14 %	27	32 %
	Validation	18	21 %	4	5 %	10	12 %
Type System Definition	Directives	17	20 %	10	12 %	10	12 %
	Enums	26	31 %	13	15 %	13	15 %
	Input Object	28	33 %	7	8 %	15	18 %
	Interfaces	31	37 %	11	13 %	13	15 %
	Objects	59	70 %	50	60 %	62	74 %
	Scalars	39	46 %	47	56 %	52	62 %
	Schemas	73	87 %	46	55 %	66	79 %
	Unions	26	31 %	8	10 %	8	10 %
Executable Definition	Fragments	20	24 %	13	15 %	9	11 %
	Mutations	63	75 %	24	29 %	42	50 %
	Query	80	95 %	65	77 %	75	89 %
	Subscription	31	37 %	6	7 %	5	6 %

Fig. 26. GraphQL components in publications.

shows the specific mapping of GraphQL components, their frequencies, and percentage over the total number of publications.

On the one hand, the most mentioned, exemplified, and used components in the publications are queries, schemas, objects, scalars, and mutations, confirming that these components are fundamental to building GraphQL APIs. On the other hand, the least studied components are directives, validation, and subscriptions, revealing research gaps in these components. The lack of studies on subscriptions caught our attention, because it is a primary function of the GraphQL service.

Public API used: We identified the trend of using public APIs in the publications; in this sense, we found 15 public GraphQL or REST APIs used in 15 publications (17.86%) of the SMS. The public GraphQL APIs are GitHub in studies [S24, S28, S30, S32, S47, S56, S73], Apollo Demo and Smalltalk GraphQL Demo in study [S39], and Yelp in studies [S39, S47]. The REST APIs used are GitHub in studies [S01, S40, S56]; APIs.guru in studies [S01, S13, S32]; arXiv in study [S24]; Kentico Cloud's Delivery in study [S22]; IBM Watson Language Translator in study [S13]; JAX-RS, jBPM, and KIE in study [S45]; Libraries.io in study [S01]; Toggl and Toggl Report in study [S51]; and Spotify in study [S35]. We note that GitHub's REST API and GraphQL API are the most widely used, followed by APIs.guru; they have become popular in the scientific community as a source of information for empirical studies.

4.7 RQ7: What Are the Frontiers of the Research on GraphQL?

In this question, we discuss the most relevant findings and open questions that the primary studies expose in their results, discussion, conclusions, and future work segments to summarize the state of the art of GraphQL research in four blocks: *(i) Advantages*, where we describe the studied benefits and recommended applicability of using GraphQL; *(ii) Disadvantages*, where we expose the shortcomings that prevent the application of GraphQL in specific situations; *(iii) Limitations*,

Table 13. GraphQL Advantages, Disadvantages, and Limitations Identified in Primary Studies

	Context	Dimensions	Publications
Advantages	Comparison with REST and SOAP	Performance	S03, S09, S12, S18, S22, S29, S31, S34, S37, S43, S45, S50, S55, S66, S67, S75, S76, S78
		Response size	S09, S18, S29, S31, S34, S50, S66, S77, S78
		Dynamic query	S03, S04, S18, S22, S24, S55, S75, S76
		Overload	S18, S67, S75
	SE Process	Versioning	S03, S22, S24
		Interoperability	S04, S22, S56
		Schemes	S24
Disadvantages	Comparison with REST and SOAP	Performance	S34, S77, S78, S80
Limitations	Data management	Schemes	S10, S24, S75
		Security	S22, S75
		Cache	S04, S24
		Mutations	S75
	SE Process	Interoperability	S04, S75

that lists the identified areas where GraphQL does not provide adequate functionality compared to other alternatives; and *(iv) Challenges*, where we discuss the main open questions that reveal several areas of future work. Table 13 shows the classification of publications by the advantages, disadvantages, and limitations of GraphQL and their main studied dimensions.

Advantages of GraphQL. Most publications have compared GraphQL to the industry standard REST paradigm, and also to SOAP in practical use cases. In the "performance" dimension, we abstracted the response time, efficiency, throughput, over-fetching, and under-fetching criteria exposed in the publications. The findings report that GraphQL is faster than REST by 0.02 times for simple, one endpoint queries [S12], 16 times in complex queries (four endpoints with 1,000 records) [S34], and up to 187 times for complex queries (over five endpoints) [S67]. Regarding SOAP, study [S37] reports that GraphQL improves the "response time" up to 2.49 times on queries with 1,500 concurrent users and 5.03 times the "throughput" on queries with 50 concurrent users. The dimension "response size" of GraphQL queries shows that they are smaller than REST, between 0.21 times for simple queries [S50] up to 38 times [S34] on complex queries (5 endpoints), while study [S77] reports that GraphQL reduces up to 3.9 times compared to SOAP on queries of 3 nested tables with 100 records. On the "dynamic query" dimension, studies report that GraphQL offers better flexibility when obtaining specific fields from queries and that queries can be dynamically composed by the client and interpreted by the server. The studies classified under the "overload" dimension indicate that the number of query requests is reduced between 17 [S18] and 1,002 [S67] REST requests to 1 request in GraphQL with 1,000 records.

In the context of the *SE Process*, the "versioning" dimension shows that GraphQL eliminates the need for increasing version numbers in APIs [S24], which increases maintainability [S03], and ease of versioning compared to REST APIs [S22]. In the "interoperability" dimension analyzed, studies show that GraphQL increases the reusability of operations [S04], frontend developers need less coordination with the backend [S22], and have better syntax for reading code and less effort to specify parameters with support in tools such as GraphiQL [S56] concluding, for example, that a novice developer spent 63% of his time in REST and 37% in GraphQL to perform query tasks. GraphQL's "schema" dimension shows that it is strongly typed and can be validated before execution, which also facilitates the combination of several APIs into one [S24].

Disadvantages of GraphQL. Studies that focus on the comparison with REST and SOAP paradigms show that in the "performance" dimension GraphQL is from 0.36 times [S34] to 2.5 times slower than REST on simple queries from an endpoint with a sample of 100,000 requests [S80]. They also show that GraphQL is slower than SOAP in similar settings, especially with simple queries made with 100 records [S77].

GraphQL Limitations. Five publications have mentioned more clearly the inherent limitations of GraphQL in *Data management*: "schemes" do not support private fields and are therefore visible to client applications [S24]; large schemes suggest a degree of complexity in their comprehensibility for implementing complex queries, and there is the probability of object name collisions [S75]; in study [S10] showed that 39.73% of schemes have at least one cycle in a sample of 2094 APIs, which could affect the effectiveness and efficiency of data handling. "Security" should be managed to avoid excessive data queries or request overloads leading to a denial of service [S22, S75]. "Cache" handling does not follow the HTTP specification, instead using a single endpoint [S24], which is corrected by implementing a cache management library, but data invalidation must be managed [S04]. Concerning "mutations," GraphQL does not support polymorphism (does not inherit input objects), fields may or may not be updated when receiving a null value, and there is a difference between input and output data formats [S75]. With respect to the *SE Process*, the interoperability between co-existing REST and GraphQL endpoints in turn limits maintainability and separation of concerns [S04, S75].

GraphQL Challenges. With respect to the *comparison with REST and SOAP* area, the studies identify the need to conduct extensive studies of the quality characteristics in dynamic, parallel queries [S02, S43] in different networks, programming languages, query languages, and databases [S29, S34, S77, S78, S80] but subject to a scale and data complexity according to the industrial context, to establish the appropriate conditions for the use of these architectures. Focusing on *data management*, we highlight the following key challenges: promote studies of hybrid architectures between REST and GraphQL [S12, S13] to reuse existing REST APIs or as a transition from REST to GraphQL; analyze in-depth mutation and subscription operations, which have not yet been thoroughly studied [S03]; and analyze existing caching and security practices [S17] to realize the impact of their applicability, identify the best existing techniques or establish new proposals. As technical challenges related to the *SE process*, studies propose to build code generative tools for advanced elements such as paging or nested queries [S13], analyze how practitioners use GraphQL in real systems [S56], and analyze the impact of transitioning architectures such as REST or SOAP [S74], with the aim of improving the software development process. There are also additional challenges on the applicability of GraphQL in cutting-edge areas, such as devising a standardized approach and the semantics of GraphQL in the context of knowledge graphs [S06] or its application in IoT use cases [S14].

5 ANALYSIS AND DISCUSSION

In this section, we analyze and discuss the different relationships between the results obtained in the SMS. Before starting, we clarify that the general classification results of the publications (topic-independent classification) are in RQ1, RQ2, RQ3, RQ4, RQ5, and the specific classification about the GraphQL paradigm (topic-specific classification) in RQ6 and RQ7.

In RQ1 (Section 4.1), we observe that institutions from Europe are leaders in the study about GraphQL and have the majority (60.74%) of publications, followed with a significant distance by the other continents. However, we note that there is not yet any scientific community specialized in this topic, although we highlight the contributions of Linkoping University.

In RQ2 (Section 4.2), we identified that 46.43% of the publications are conference papers, followed by gray literature, journal articles, and workshops. Figure 27 shows the relationship of RQ2,

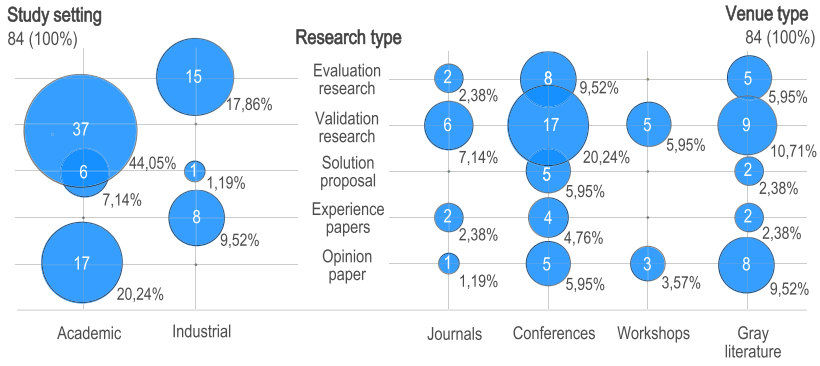


Fig. 27. Publications by study setting, research type, and venue type.

RQ3, and RQ5 results to analyze the research types contributed by the venues; we conclude that including the gray literature in the SMS was a correct decision, because 53.84% of its production provided evaluation and validation research. Although GraphQL is a young topic, we highlight publications in top-level conferences such as WWW (Class 1: A++) and ESEC/FSE (Class 1: A+) and Quartile 1 journals such as Molecular Biology, Cheminformatics, and IT Professional.

In RQ3 (Section 4.3), we identified that most of the publications are of validation or evaluation research type (61.91%), presenting a certain degree of maturity in empirical evaluation in their studies; however, only 17.86% of these studies were conducted in SE industry practice (evaluation research). Moreover, we analyzed the venue types concerning "validation" and "evaluation" research. Note that the grey literature shows 16.66% of studies with empirical evaluations even applied in the SE industry settings; this indicates that universities contribute to relevant academic studies in the GraphQL study (see Figure 27). In this sense, we discuss if it is possible to improve the maturity of empirical evidence of GraphQL studies. We find that future research can improve this maturity if researchers use the methods described in Section 3.5 and apply them in the industry or government setting (i.e., SE practice).

In RQ4 (Section 4.4), we find a growth of studies in quantity and quality. Figure 28 shows the relationship of the results of RQ2, RQ3, and RQ4; we observe that from 2019 appear works from journals that generally have more rigor for publication, as well as the quality of the empirical evidence exposed in "validation" and "evaluation" research. These results corroborate the motivation for conducting the present SMS.

In RQ5 (Section 4.5), on the one hand, we first analyzed the study setting concerning the research type. We observed that most of the publications are of validation research type in academia (see Figure 27); this suggests that researchers did their studies in synthetic contexts without applying them in SE practice (i.e., industry or government). In this sense, we found that another key point to improve the quality of GraphQL research is to apply the studies in industry and government settings. On the other hand, we note that the publications conducted their study in 7 of the 15 SWEBOK knowledge areas (see Figure 22), opening a research opportunity in the rest of the knowledge areas described in Section 3.5.

In RQ6 (Section 4.6), we answer how the scientific community studied the GraphQL paradigm through the following classification metrics: technical contributions, their types and domains, GraphQL components, and public APIs used in publications. Figure 29 shows the relationship between RQ3 and RQ6.

First, Figure 23 shows that the researchers turned to study the domain provided by the GraphQL service (72.73%); we think this is normal behavior, because the GraphQL service must first be

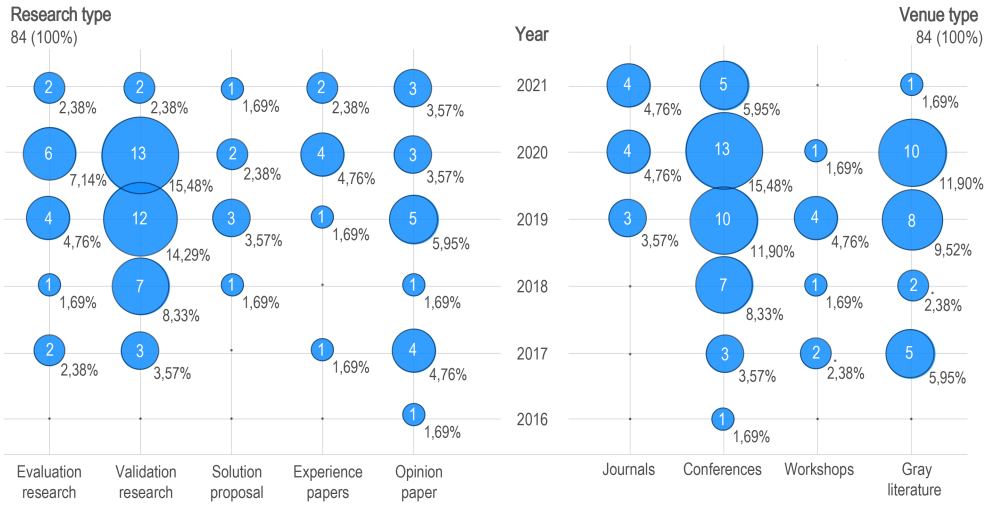


Fig. 28. Publications by research type, year, and venue type.

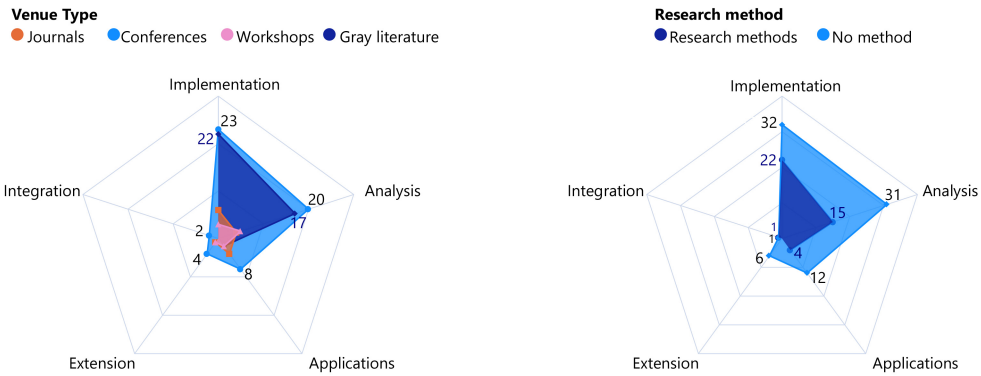


Fig. 29. Publications by contribution type, venue type, and research method.

implemented before its consumption; in this sense, we identify a study opportunity on usability, ease of learning, and consumption performance of the GraphQL service. Second, the studies' most frequent contribution types are implementation and GraphQL analysis (see Figure 29), published with similar frequency in conference papers and gray literature; in this sense, we note that implementations can improve their quality by validating or evaluating their studies using the research methods described in Section 3.5. Third, in the "analysis" contribution type, the comparisons between REST and GraphQL are highlighted (see Table 12), clearly showing that the community studied GraphQL as an alternative to REST for implementing web APIs. Fourth, we mapped the components of the GraphQL paradigm that were mentioned, used, and exemplified in the publications. In a general way, we observed that the most studied components are queries, schemas, objects, and scalar types; we conclude that this behavior is because they are fundamental and necessary components to implement GraphQL APIs (see Figure 26). In this sense, we asked if there are components that should be studied more. This question surprised us, because most components are not treated in detail, including essential components such as mutations and subscriptions,

evidencing another gap in studies. Finally, we identified that the GitHub public API is the most used in the publications, identifying a tendency to implement public REST APIs and support GraphQL APIs.

Finally, in RQ7 (Section 4.7), we analyze the main findings of the publications and classify them to understand the frontiers in GraphQL research. Many studies highlight the advantages of GraphQL compared to REST (and SOAP) in features and quality metrics such as throughput, response size, dynamic query, and overload. We find that the right conditions for using REST are when APIs perform static queries with little data; in contrast, GraphQL excels when performing dynamic queries with big datasets. As a possible roadmap for future work to overcome the limitations and challenges reported in GraphQL, we propose:

(i) Strengthen the baseline conditions for using REST and GraphQL to build efficient applications by increasing studies of experimental design complexity and data quantity scale (similar to those used in industrial environments). Also, adding different features such as quality characteristics (e.g., usability, functional adequacy), GraphQL operations (mutations and subscriptions), network conditions (e.g., bandwidth, concurrency), programming languages (backend, frontend), query languages (e.g., falcor, SPARQL), databases (relational and non-relational), and applied on various platforms such as mobile and IoT.

(ii) Establish a baseline of existing best practices and encourage proposals for the management of caching, security in GraphQL APIs, and control of the inherent problems of GraphQL schemes and mutations reported in Section 4.7.

(iii) Establish proposals that promote the governance of the software engineering process in developing GraphQL APIs and hybrid architectures between REST and GraphQL, such as code generation tools for advanced GraphQL components (e.g., pagination and nested queries), testing GraphQL APIs [32], or the management of GraphQL API service level agreements [35, 39].

(iv) Non-Functional aspects such as *limitations* or *pricing* represent a key element for API practitioners [13, 16] and are widely used in the RESTful API market [15]; in contrast, this study shows a lack of approaches that address those concerns for GraphQL, which prevents the consolidation of an open market of GraphQL-based services as it has been proposed in similar contexts [17, 18, 47].

6 CONCLUSIONS

GraphQL is a novel approach to APIs development that is gaining traction and interest from both researchers and practitioners. To offer a global perspective on this field, in this work, we first introduce a conceptual framework to describe the so-called GraphQL paradigm from its formal specification, illustrating and exemplifying its components to serve as the basis to acquire a deep understanding of the GraphQL paradigm and relate the different elements studied. Then, as the main focus of our work, we conducted a SMS of the literature to show an overview of the research and identify trends and gaps in the GraphQL field. The SMS answered seven research questions and classified the studies into general and specific areas about GraphQL. In particular, our study focused on finding out who, where, when, what, and why GraphQL was researched, as well as contextualizing the specific areas of research with respect to the GraphQL paradigm we introduced. The results of our study confirm that in spite of its growing acceptance as an alternative for API development, there is not a widely established scientific community around GraphQL. Even though there is a trend of publishing in high-quality venues, there are still shortcomings in current studies, especially in terms of the maturity of empirical evidences, validation in realistic use cases, and the evaluation of additional quality characteristics and underused features of GraphQL. Furthermore,

we also detected research opportunities in other areas related to best practices and proposals for the development and governance of GraphQL-based systems, including aspects like security, testing, and service level agreements, among others. In summary, even though the field is young and needs to mature in some aspects, this study will help researchers get an overview of the topics investigated and directions for future GraphQL research.

VERIFIABILITY

For verifiability purposes, we published the following Supplementary Materials in Reference [45]: (1) Study identification process dataset, (2) SMS primary study dataset, and (3) SMS primary study mapping dataset.

REFERENCES

[1] H. Arksey and L. O'Malley. 2005. Scoping studies: Towards a methodological framework. *Int. J. Soc. Res. Methodol.: Theory Pract.* 8, 1 (2005), 19–32. https://doi.org/10.1080/1364557032000119616

[2] S. Baškarada, V. Nguyen, and A. Koronios. 2018. Architecting microservices: Practical opportunities and challenges. *J. Comput. Info. Syst.* 60, 5 (2018), 1–9. https://doi.org/10.1080/08874417.2018.1520056

[3] P. Bourque and R. E. Fairley. 2014. *SWEBOK v.3—Guide to the Software Engineering—Body of Knowledge.* (3rd ed.). IEEE, 346 pages. https://doi.org/10.1234/12345678

[4] R. Budde, K. Kautz, K. Kuhlenkamp, and H. Züllighoven. 1992. *Prototyping An Approach to Evolutionary System Development.* Springer-Verlag, Berlin. XII, 205 pages. https://doi.org/10.1007/978-3-642-76820-0

[5] Clarivate. 2021. Journal Citation Reports: Reference Guide. Retrieved from https://clarivate.com/webofsciencegroup/wp-content/uploads/sites/2/2021/06/JCR_2021_Reference_Guide.pdf.

[6] Clarivate. 2021. Quartiles in JCR on the InCites Platform. Retrieved from https://help.incites.clarivate.com/incitesLiveJCR/9053-TRS.

[7] N. Condori-Fernandez, M. Daneva, K. Sikkel, R. Wieringa, O. Dieste, and O. Pastor. 2009. A systematic mapping study on empirical evaluation of software requirements specifications techniques. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09)* 1 (2009), 502–505. https://doi.org/10.1109/ESEM.2009.5314232

[8] P. Di Francesco, P. Lago, and I. Malavolta. 2019. Architecting with microservices: A systematic mapping study. *J. Syst. Softw.* 150 (2019), 77–97. https://doi.org/10.1016/j.jss.2019.01.001

[9] T. Dyba, T. Dingsoyr, and G. Hanssen. 2007. Applying systematic reviews to diverse study types: An experience report. In *Proceedings of the (ESEM'07).* IEEE, 225–234. https://doi.org/10.1109/ESEM.2007.59

[10] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. 2008. Selecting empirical methods for software engineering research. *Guide Adv. Empir. Softw. Eng.* 1 (2008), 285–311. https://doi.org/10.1007/978-1-84800-044-5_11

[11] Elsevier. 2017. FAQs—Journal Metrics—Scopus.com. Retrieved from https://journalmetrics.scopus.com/index.php/Faqs.

[12] Inc. Facebook. 2016. GraphQL | A query language for your API. Retrieved from https://graphql.org/.

[13] R. Fresno-Aranda, P. Fernández, A. Durán, and A. Ruiz-Cortés. 2022. Semi-automated capacity analysis of limitation-aware microservices architectures. In *Proceedings of the 19th International Conference on the Economics of Grids, Clouds, Systems, and Services (GECON'22).* Springer, In press.

[14] A. Galindo, D. Benavides, P. Trinidad, A. Gutiérrez-Fernández, and A. Ruiz-Cortés. 2019. Automated analysis of feature models: Quo vadis? *Computing* 101, 5 (2019), 387–433. https://doi.org/10.1007/s00607-018-0646-1

[15] A. Gamez-Diaz, P. Fernandez, and A. Ruiz-Cortes. 2017. An analysis of RESTful APIs offerings in the industry. In *Proceedings of the International Conference on Service-Oriented Computing.* Springer, Cham, Germany, 589–604. https://doi.org/10.1109/SCC.2016.17

[16] A. Gamez-Diaz, P. Fernandez, A. Ruiz-Cortés, P. J. Molina, N. Kolekar, P. Bhogill, M. Mohaan, and F. Méndez. 2019. The role of limitations and SLAs in the API industry. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* ACM, New York, NY, 1006–1014. https://doi.org/10.1145/3338906.3340445

[17] J. M. García, O. Martin-Diaz, P. Fernandez, C. Müller, and A. Ruiz-Cortes. 2021. A flexible billing life cycle for cloud services using augmented customer agreements. *IEEE Access* 9 (2021), 44374–44389. https://doi.org/10.1109/ACCESS.2021.3066443

[18] J. M. García, O. Martin-Diaz, P. Fernandez, A. Ruiz-Cortes, and M. Toro. 2017. Automated analysis of cloud offerings for optimal service provisioning. In *Proceedings of the International Conference on Service-Oriented Computing.* Springer, Cham, Germany, 331–339. https://doi.org/10.1007/978-3-319-69035-3_23

[19] M. Genero, A. Cruz-Lemus, and M. Piattini. 2014. *Métodos de Investigación en ingeniería del Software*. Ra-Ma, Bogota.
[20] Google. 2021. FAQ about Trends. Retrieved from https://support.google.com/trends/answer/4365533?hl=en&ref_topic=6248052.
[21] C. Guevara-Vega, B. Bernardez, A. Duran, A. Quina-Mera, M. Cruz, and A. Ruiz-Cortes. 2021. Empirical strategies in software engineering research: A literature survey. In *Proceedings of the 2nd International Conference on Information Systems and Software Technologies (ICI2ST'21)*. IEEE, New York, NY, 120–127. https://doi.org/10.1109/ICI2ST51859.2021.00025
[22] H. He and A. K. Singh. 2008. Graphs-at-a-time: Query language and access methods for graph databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*. ACM, 405–417. https://doi.org/10.1145/1376616.1376660
[23] T. Hunter II. 2017. *Advanced Microservices: A Hands-on Approach to Microservice Infrastructure and Tooling*. Apress, 193 pages. https://doi.org/10.1007/978-1-4842-2887-6
[24] R. Kipling. 1902. *Just So Stories*. MacMillan, London.
[25] B. Kitchenham and P. Brereton. 2013. A systematic review of systematic review process research in software engineering. *Info. Softw. Technol.* 55, 12 (2013), 2049–2075. https://doi.org/10.1016/j.infsof.2013.07.010
[26] M. Kuhrmann, D. M. Fernández, and M. Daneva. 2017. On the pragmatic design of literature studies in software engineering: An experience-based guideline. *Empir. Softw. Eng.* 22, 6 (2017), 2852–2891. https://doi.org/10.1007/s10664-016-9492-y
[27] M. A. Laguna and Y. Crespo. 2013. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Sci. Comput. Program.* 78, 8 (2013), 1010–1034. https://doi.org/10.1016/j.scico.2012.05.003
[28] B. Lee. 2018. Introducing the GraphQL Foundation—Lee Byron—Medium. Retrieved from https://medium.com/@leeb/introducing-the-graphql-foundation-3235d8186d6d.
[29] S. Li, H. Zhang, Z. Jia, Z. Li, C. Zhang, J. Li, Q. Gao, J. Ge, and Z. Shan. 2019. A dataflow-driven approach to identifying microservices from monolithic applications. *J. Syst. Softw.* 157 (2019), 16. https://doi.org/10.1016/j.jss.2019.07.008
[30] C. Y. Lin, T. Abdel-Hamid, and J. S. Sherif. 1997. Software-engineering process simulation model (SEPS). *J. Syst. Softw.* 38, 3 (1997), 263–277. https://doi.org/10.1016/S0164-1212(96)00156-2
[31] L. Maloney. 2020. How to Calculate Percent Variation. Retrieved from https://sciencing.com/calculate-percent-variation-7538781.html.
[32] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés. 2022. Online testing of RESTful APIs: Promises and challenges. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'22)*. ACM, In press.
[33] Peter M. Mell and Timothy Grance. 2011. *The NIST Definition of Cloud Computing*. Technical Report. National Institute of Standards and Technology, Gaithersburg.
[34] S. Mukhiya, F. Rabbi, V. Pun, A. Rutle, and Y. Lamo. 2019. A graphql approach to healthcare information exchange with hl7 fhir. In *Proceedings of the 9th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH'19)*, Vol. 160. Elsevier B.V., Coimbra, 338–345. https://doi.org/10.1016/j.procs.2019.11.082
[35] C. Müller, A. M. Gutierrez, O. Martin-Diaz, M. Resinas, P. Fernandez, and A. Ruiz-Cortes. 2014. Towards a formal specification of slas with compensations. In *Proceedings of the OTM Confederated International Conferences*. Springer, Berlin, 295–312. https://doi.org/10.1007/978-3-662-45563-0_17
[36] O. Emily and The Linux Foundation. 2018. The Linux Foundation Announces Intent to Form New Foundation to Support GraphQL—The Linux Foundation. Retrieved from https://www.linuxfoundation.org/press-release/intent_to_form_graphql.
[37] Group of Italian Professors of Computer Engineering, Group of Italian Professors of Computer Science, and Spanish Computer-Science Society. 2018. The GII-GRIN-SCIE (GGS) Conference Rating. Retrieved from http://scie.lcc.uma.es/.
[38] A. Paez. 2017. Grey literature: An important resource in systematic reviews. *J. Evid.-based Med.* 10, May (2017), 1–8. https://doi.org/10.1111/jebm.12265
[39] J. A. Parejo, P. Fernandez, A. Ruiz-Cortes, and J. M. García. 2008. Slaws: Towards a conceptual architecture for sla enforcement. In *Proceedings of the IEEE Congress on Services*. IEEE, 322–328. https://doi.org/10.1109/SERVICES-1.2008.80
[40] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. 2008. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08)*. BCS Learning and Development, 11. https://doi.org/10.14236/ewic/ease2008.8
[41] K. Petersen and C. Gencel. 2013. Worldviews, research methods, and their relationship to validity in empirical software engineering research. In *Proceedings of the Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement (IWSM-MENSURA'13)*. IEEE, 81–89. https://doi.org/10.1109/IWSM-Mensura.2013.22

[42] K. Petersen, S. Vakkalanka, and L. Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Info. Softw. Technol.* 64 (2015), 1–18. https://doi.org/10.1016/j.infsof.2015.03.007
[43] Prisma. 2017. Execution | GraphQL. Retrieved from https://graphql.org/learn/execution/.
[44] Prisma. 2017. GraphQL vs REST—A comparison. Retrieved from https://www.howtographql.com/basics/1-graphql-is-the-better-rest/.
[45] A. Quiña-Mera, P. Fernandez, J. M. García, and A. Ruiz-Cortés. 2021. GraphQL: A Systematic Mapping Study—Supplemental Material. Retrieved from https://zenodo.org/record/6036802#.YyMq-3bMKUk. https://doi.org/10.5281/zenodo.5155990
[46] M. Raza, F. Khadeer, O. Khadeer, and M. Zhao. 2019. A comparative analysis of machine learning models for quality pillar assessment of SaaS services by multi-class text classification of users' reviews. *Future Gen. Comput. Syst.* 101 (2019), 341–371. https://doi.org/10.1016/j.future.2019.06.022
[47] M. Resinas, P. Fernandez, and R. Corchuelo. 2010. Automatic service agreement negotiators in open commerce environments. *Int. J. Electr. Comm.* 14, 3 (2010), 93–128. https://doi.org/10.2753/JEC1086-4415140305
[48] SCImago. 2012. SJR—About Us. Retrieved from https://www.scimagojr.com/aboutus.php.
[49] SCImago. 2019. SJR—Help. Retrieved from https://www.scimagojr.com/help.php.
[50] P. Seifer, J. Härtel, M. Leinberger, R. Lämmel, and S. Staab. 2019. Empirical study on the usage of graph query languages in open source Java projects. In *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering (SLE'19)*. ACM, 152–166. https://doi.org/10.1145/3357766.3359541
[51] Facebook Open Source. 2015. Schemas and Types | GraphQL. Retrieved from https://graphql.org/learn/schema/.
[52] The GraphQL Foundation. 2018. GraphQL. Retrieved from https://graphql.github.io/graphql-spec/June2018/.
[53] J. Thönes. 2015. Microservices. *IEEE Software* 32 (2015), 4. https://doi.org/10.1109/MS.2015.11
[54] W. T. Tsai, X. Y. Bai, and Y. Huang. 2014. Software-as-a-service (SaaS): Perspectives and challenges. *Sci. China Info. Sci.* 57, 5 (2014), 1–15. https://doi.org/10.1007/s11432-013-5050-z
[55] M. Vogel, S. Weber, and C. Zirpins. 2018. Experiences on migrating RESTful web services to GraphQL. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'17)*. Springer Verlag, 283–295. https://doi.org/10.1007/978-3-319-91764-1_23
[56] S. Wang, I. Keivanloo, and Y. Zou. 2014. How do developers react to RESTful API evolution? *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8831 (2014), 245–259. https://doi.org/10.1007/978-3-662-45391-9_17
[57] R. Wieringa, N. Maiden, N. Mead, and C. Rolland. 2006. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. *Require. Eng.* 11, 1 (2006), 102–107. https://doi.org/10.1007/s00766-005-0021-6
[58] C. Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. ACM, London, 10. https://doi.org/10.1145/2601248.2601268
[59] C. Wohlin, P. Runeson, P. A. da Mota Silveira Neto, E. Engström, I. do Carmo Machado, and E. S. de Almeida. 2013. On the reliability of mapping studies in software engineering. *J. Syst. Softw.* 86, 10 (2013), 2594–2610. https://doi.org/10.1016/j.jss.2013.04.076
[60] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering* (1st ed.). Springer-Verlag, Berlin. XXIV, 236 pages. https://doi.org/10.1007/978-3-642-29044-2
[61] O. Zimmermann. 2017. Microservices tenets: Agile approach to service development and deployment. *Comput. Sci. Res. Dev.* 32, 3–4 (2017), 301–310. https://doi.org/10.1007/s00450-016-0337-0

Received 3 August 2021; revised 19 August 2022; accepted 23 August 2022