JsonCurer: Data Quality Management for JSON Based on an Aggregated Schema

Kai Xiong, Xinyi Xu, Siwei Fu, Di Weng, Yongheng Wang, and Yingcai Wu

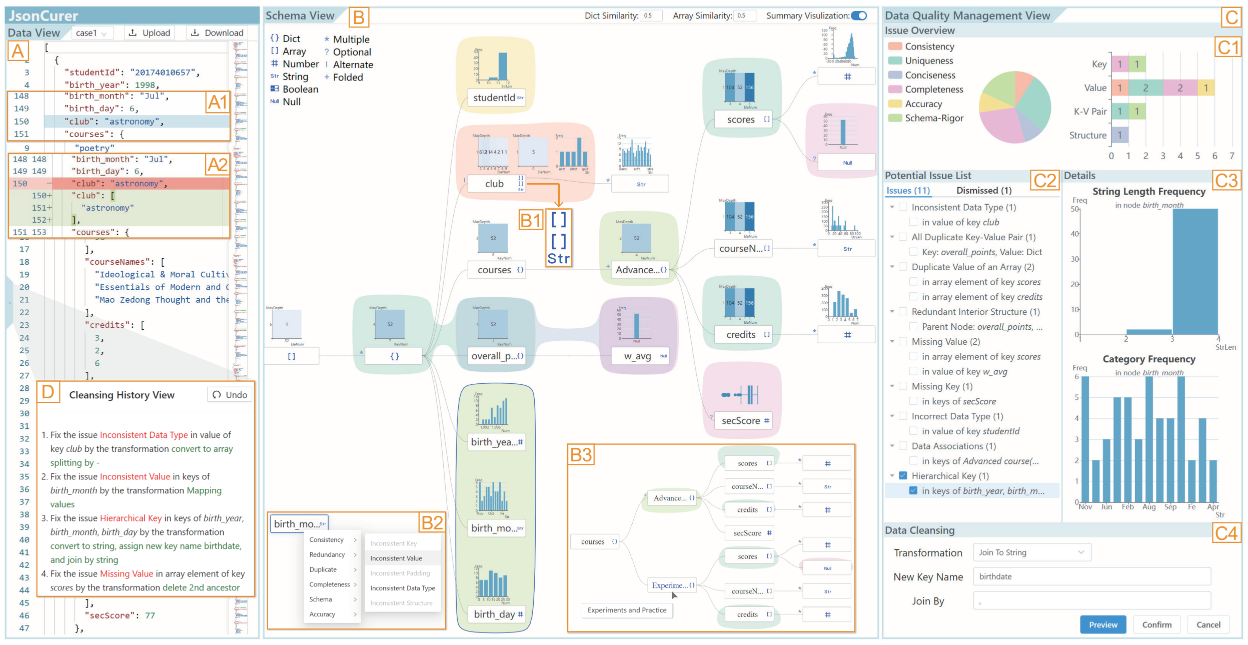


Fig. 1: The interface of JsonCurer. (A) The Data View enables users to upload raw JSON data; (B) The Schema View presents an aggregated schema tree rendered with summary visualizations and Bubble Sets; (C) The Data Quality Management View illustrates the overviews and details of potential quality issues and recommends proper transformations to fix them; (D) The Cleansing History View logs all transformations performed throughout the process of improving the quality of JSON data.

Abstract—High-quality data is critical to deriving useful and reliable information. However, real-world data often contains quality issues undermining the value of the derived information. Most existing research on data quality management focuses on tabular data, leaving semi-structured data under-exploited. Due to the schema-less and hierarchical features of semi-structured data, discovering and fixing quality issues is challenging and time-consuming. To address the challenge, this paper presents JsonCurer, an interactive visualization system to assist with data quality management in the context of JSON data. To have an overview of quality issues, we first construct a taxonomy based on interviews with data practitioners and a review of 119 real-world JSON files. Then we highlight a schema visualization that presents structural information, statistical features, and quality issues of JSON data. Based on a similarity-based aggregation technique, the visualization depicts the entire JSON data with a concise tree, where summary visualizations are given above each node, and quality issues are illustrated using Bubble Sets across nodes. We evaluate the effectiveness and usability of JsonCurer with two case studies. One is in the domain of data analysis while the other concerns quality assurance in MongoDB documents. The source code of JsonCurer is available under the Apache License 2.0 at https://github.com/changevis/JsonCurer.

Index Terms—Data quality, JSON, Schema aggregation, Visualization

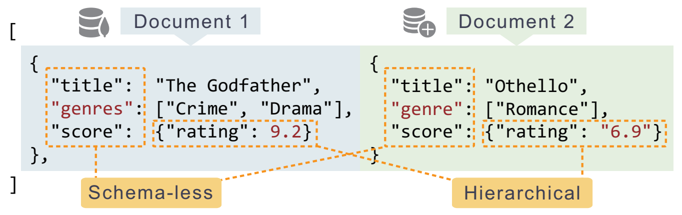
1 Introduction

- *Kai Xiong and Yingcai Wu are with the State Key Lab of CAD&CG, Zhejiang University, Hangzhou, China. E-mail: {kaixiong, ycwu}@zju.edu.cn.*
- *Xinyi Xu and Di Weng are with the School of Software Technology, Zhejiang University, Ningbo, China. E-mail: {xinyixu, dweng}@zju.edu.cn.*
- *Siwei Fu and Yongheng Wang are with the Zhejiang Lab, Hangzhou, China. E-mail: fusiwei339@gmail.com, wangyh@zhejianglab.com.*
- *Yingcai Wu and Siwei Fu are the co-corresponding authors.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

Nowadays, with the rapid growth of data, a myriad of human activities (e.g., data analytics and decision making) and database applications (e.g., social media websites and customer relationship management systems) rely on it to derive useful information [62]. The meaningfulness, reliability, and interpretability of the derived information highly depend on the quality of the data [44]. However, data sets regularly encompass quality issues, such as missing, inconsistent, or duplicate values, which stem from various sources, including user entry errors, uncoordinated data integration, and poorly applied coding standards [53], resulting in a huge annual economic loss of millions of dollars [65]. Therefore, it is critical to discover and fix quality issues before utilizing data.

Currently, some software applications [11, 14, 15] and research papers [21, 53] have been devoted to assisting users in assessing and improving the quality of tabular datasets, leaving semi-structured data under-exploited [84]. Semi-structured data, e.g., JavaScript Object



```
[
  {
    "title":  "The Godfather",
    "genres": ["Crime", "Drama"],
    "score":  {"rating": 9.2}
  },
  {
    "title":  "Othello",
    "genre":  ["Romance"],
    "score":  {"rating": "6.9"}
  }
]
```

Fig. 2: An example of a MongoDB collection highlighting two quality issues (i.e., inconsistent keys and inconsistent data types) associated with the schema-less and hierarchical features of semi-structured data.

Notation (JSON), XML, and NoSQL database, is ubiquitous and plays a progressively more important role with the rapid development of the Internet [80]. Consequently, ensuring its high quality is vital in real-world scenarios. For example, in exploratory data analysis, one has to first solve quality issues to obtain valuable and reliable results. Besides, in NoSQL database quality assurance, data stewards must assess and ensure that the quality of data complies with predefined needs or standards. Chu et al. [31] acknowledged that quality issues for semi-structured data remain a challenge in data cleansing research.

There are two salient features that distinguish semi-structured data from structured data. First, semi-structured data is **schema-less** [22, 48, 89], as it is not restricted to a fixed or rigid schema defined in advance, allowing keys to be added, changed, or deleted at any time. For example, in Fig. 2, Document 1 uses the key 'genres' to represent the movie genres, while Document 2 uses 'genre' for the same purpose. Though flexible, this inconsistency impedes data comprehension and transformation. Second, unlike structured data that typically represents data in a table format, semi-structured data supports a **hierarchical** structure. Identifying quality issues that reside deep within the hierarchy requires traversal along the hierarchy. For instance, in Fig. 2, both documents utilize *dict* as the value type for the 'score' key. However, after unraveling the nested structure of the two *dicts*, Document 1 uses a numeric data type for the 'rating' value, whereas Document 2 uses a string data type. Discovering such inconsistent data types within nested structure is often a laborious process for users. Though the two features allow semi-structured data to represent a broad range of information, making it an ideal format for web services, they introduce diverse and complex data quality issues, which can be challenging and time-consuming for users to detect and resolve.

In this paper, we propose JsonCurer, an interactive visualization system designed to assist with data quality management for semi-structured data. We concretize semi-structured data into JSON, which is becoming the most common semi-structured format [41]. While our primary focus is on JSON, we believe our technique can be generalized to other semi-structured data formats, such as XML. We follow the design study methodology [78] to develop JsonCurer. Specifically, we first closely collaborate with three data practitioners to formulate design requirements, identify quality issues in JSON data, and explore possible solutions. We also collect a corpus containing 119 JSON files with quality issues and their corresponding solutions from real world scenarios to enrich the issue types. After that, we derive a taxonomy including 27 quality issues under two axes, i.e., data objects and data quality dimensions. Guided by the taxonomy, we then develop an engine to automatically detect potential issues behind data and recommend proper transformations to fix them. Next, we propose a similarity-based aggregation technique to summarize JSON schema and visualize it as a node-link tree, accompanied by summary visualizations and Bubble Sets to facilitate the comprehension of data and its quality issues. Last, we implement a prototype of JsonCurer to support iterative improvement of data quality. To evaluate the effectiveness and usability of JsonCurer, we design two case studies in different scenarios. The results show that JsonCurer mitigates incorrect analysis conclusions caused by poor quality data. Meanwhile, it assists data stewards to remove duplicate and redundant data in MongoDB. We report insights and reflections obtained from interviews with data practitioners.

In summary, our main contributions include: 1) a taxonomy of 27 types of quality issues for JSON data, 2) a similarity-based schema aggregation technique for JSON and a visual design for depicting the schema, 3) an interactive visualization system, called JsonCurer, that helps users discover and fix quality issues efficiently and conveniently, and 4) two case studies that demonstrate the effectiveness and usability of JsonCurer. We also present a sub-contribution as supplementary material, i.e., a real-world corpus containing 119 JSON files where each is annotated with quality issues and their corresponding solutions.

2 Related Work

Over the past years, a wide variety of methodologies and frameworks [23,59,64,77,87] have been proposed to guide data quality management. Despite their different characteristics and emphases, four essential activities can be extracted as possible [44], i.e., data profiling, data quality assessment, continuous quality monitoring, and data cleansing or improvement. JsonCurer involves all the aforementioned activities.

2.1 Data Profiling

Data profiling is the process of examining and analyzing a given dataset to determine its metadata that can help understand the dataset, such as data types, structures, patterns, and statistics [16, 68]. It is an essential action prior to any other activity to yield a summary view and gain insight into data [44]. Currently, various tools [11, 12, 53, 66, 73, 91, 92] leverage data profiling. For example, in the business community, OpenRefine [11] supports both text clustering and faceted browsing to aid in the identification of quality issues. Besides, in the research community, Profiler [53] integrates statistical analysis and coordinated summary visualizations to flag problematic data. However, these tools focus on tabular data and are not applicable to semi-structured data.

Some existing tools are able to extract and visualize the schema of semi-structured data. JSONDiscoverer [48] aims to facilitate the integration and composition of JSON-based Web APIs by discovering and visualizing the implicit schema of JSON documents. STEED [90] is an analytical database system that learns and extracts a schema tree for tree-structured data (such as JSON data) and provides visualizations of the schema tree and its summary statistics. JSON Crack [6] helps understand, explore, and analyze JSON structures by visualizing JSON data as graphs. SchemaDrill [81] supports presenting a summary schema representation that diminishes the complexity of JSON schemas without losing meaningful information. However, these visualization tools are not designed for quality management in JSON. For example, they cannot locate and present issues buried in a JSON file. The idea of JsonCurer is inspired by the above work while having different focuses. To be specific, our system aims to facilitate the comprehension of JSON data and the discovery of quality issues.

2.2 Data Quality Assessment and Monitoring

Data quality assessment aims to measure and estimate the quality of data and draw conclusions from it [44, 77]. It can be carried out from multiple dimensions, each representing a specific aspect of data quality [43]. Despite numerous research [24, 25, 59, 62, 76, 88] on quality dimensions, there is no consensus on a standardized set of dimensions for data quality assessment [44, 77], since the term "data quality" is context-dependent and frequently associated with the "fitness for use" principle [88]. Laranjeiro et al. [58] surveyed the state-of-the-art literature on data quality and identified frequently used dimensions, such as accuracy, completeness, and consistency. Guided by these characteristics, we consider six data quality dimensions and outline a taxonomy of quality issues based on them. Further, we design and implement the measurement and detection methods for these issues.

Data quality assessment is a cyclic process that needs to be carried out continuously as data changes. The ongoing quality assessment and the evaluation of quality improvements are described as quality monitoring [42]. According to Ehrlinger et al. [42], the implementations of continuous quality monitoring in existing work fall into three categories, i.e., 1) employing organizational control mechanisms [59], 2) constantly applying a set of test rules on data and possibly triggering alerts if these rules are violated [50], and 3) performing periodic data quality checks and visualizing the check results [17, 83]. JsonCurer supports continuous quality monitoring by triggering issue detection checks through a rule-based detection engine once data changes.

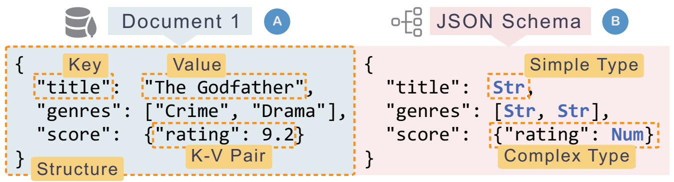
2.3 Data Cleansing

Data cleansing (aka data cleaning) deals with repairing erroneous data or data glitches [37, 44]. To improve data quality, numerous data cleansing techniques have been proposed so far. These techniques cover various aspects of data cleansing, such as interfaces [11, 15, 30, 36, 61, 73], new abstractions [32, 46, 86], crowdsourcing techniques [33, 71, 82], and approaches for scalability [55, 79, 85]. In conjunction with data cleansing, it is often mentioned that data transformations are required to support any changes in the content, representation, or structure of data [72]. Kandel et al. [52] investigated the challenges and opportunities in fixing quality issues and discussed several research directions, comprising how suitable visual encodings can help reveal erroneous data and how interactive visualizations can promote data cleansing and transformation. Kasica et al. [54] structured a concise and actionable framework for multi-table transformation. However, a preponderance of the above data cleansing and transformation work is targeted at structured data.

Some tools are designed for cleansing or transforming JSON data. For example, Data Chameleon [84] is a framework to support advanced transformations on complex semi-structured data through a Domain-Specific Language (DSL). However, it does not support detecting quality issues in data. Besides, there is a learning curve for novices to write code using DSL. Some commercial tools, such as JSON Formatter & Validator [7], JSON Cleaner [34], and JSONFormatter.io [8], claim that they help clean, debug, and beautify JSON data and are capable of detecting and fixing errors in data. However, the errors they can detect and fix are limited to JSON format issues, such as duplicate keys, trailing commas, missing/incorrect quotes, etc. On the contrary, JsonCurer supports detecting and fixing both structural and value-level quality issues under six quality dimensions.

3 Terminology and Background

3.1 Related Concepts



```
{
  "title":  "The Godfather",
  "genres": ["Crime", "Drama"],
  "score":  {"rating": 9.2}
}
```

```
{
  "title":  Str,
  "genres": [Str, Str],
  "score":  {"rating": Num}
}
```

Fig. 3: We utilize Document 1 annotated with four data objects (A) and its corresponding schema marked with two categories of data types (B) to illustrate related concepts of JSON.

To ease the understanding the rest of the paper, we describe a set of concepts related to JSON. 1) *JSON Data Types.* JSON includes six data types where *string*, *number*, *boolean*, and *null* are *simple types*, and *array* and *dict* are *complex types* that support nested structures. 2) *JSON Data Objects.* To categorize data quality issues, we distinguish four data objects (Fig. 3A) for JSON data according to different granularity levels: *key* and *value* are basic objects while *key-value pair* (*k-v pair*) and *structure* are composite objects. 3) *Hierarchy Path, Level, and Depth.* The *hierarchy path* of a value in JSON data is a list of indicators to locate it and its *hierarchy level* is the number of these indicators. For example, in Fig. 3A, we can access the value 9.2 through the keys 'score' and 'rating'. Thus, its hierarchy path is `['scores','rating']` and its hierarchy level is two. Particularly, Document 1 itself is a *dict* value whose hierarchy level is zero (as no indicators are needed to locate it), and we regard it as the *root* of the JSON data. Further, the *hierarchy depth* of a value is the maximum hierarchy level of its child values, and the hierarchy depth of Document 1 is two. 4) *JSON Schema* refers to a declarative language for annotating and validating JSON files. In this paper, we define a plain JSON schema as the combination of structural (i.e., how the data is structured) and type (i.e., what the data types of the values are) properties [22] (Fig. 3B).

3.2 Requirement Analysis

Over the past year, we closely collaborated with two data analysts and one data steward in a national research lab. Both analysts have over five years of experience in analyzing JSON data, whereas the data steward has been engaged in ensuring the quality of JSON data in MongoDB for four years. Initially we attempted to make sense of their workflow in handling JSON data. The analysts pointed out that they usually utilize libraries (e.g., flatten_json and Pandas) in Python or flattening tools (e.g., [4, 9]) to convert JSON data into tabular data for analysis. However, as one analyst mentioned, *"for complex JSON data (such as data with deep hierarchy or inconsistent schema), the (converted) result may contain lots of empty or redundant data."* He added, *"once converted, the hierarchical information of the original JSON data would be difficult to explore."* These hinder them from discovering and fixing quality issues, especially those related to the structure, hidden in the original JSON data. As for the data steward, he routinely leverages a profiling tool developed by his department to understand data in MongoDB, and discover quality issues by checking with handwritten rules. He complained that *"these visualizations (provided by the profiling tool) are not effective enough to discover a wide range of quality issues"* and *"due to different data sources and constant changes in requirements, I have to update or rewrite rules frequently."* These pain points motivated us to design JsonCurer.

To this end, we closely followed the nested model for visualization design and validation [67]. Specifically, we first interviewed the three practitioners to understand: 1) *what quality issues have been encountered in their daily work?* 2) *how do they discover these issues?* and 3) *what transformations can be employed to fix them?* Here, we define quality issues as those hinder data analysis or fail to meet standards or needs. Thereafter, we iteratively extracted the requirements that guide the system development by holding weekly discussions for nearly three months. During the development of the prototype, the practitioners were invited to evaluate our design from different aspects, such as intuitiveness and practicability. Finally, we concluded our iteration with the following requirements.

R1: Reveal JSON schema. The prerequisite for assessing data quality and discovering issues is understanding data. JSON schema is a powerful instrument for grasping and validating the structure of JSON data [40]. As the data steward noted, *"Data schema can help me quickly know what the data describe and discover its structural issues."* Therefore, our system should support extracting and portraying JSON schema.

R2: Present summaries of data. Apart from JSON schema, data practitioners also need to have a grip on the detailed data for discovering non-structural issues, such as outliers. However, direct inspection of raw data is labor-intensive. Data practitioners expect some summaries of data to facilitate exploring the detailed data.

R3: Detect potential quality issues. Since there may be miscellaneous issues hidden in data, it is hard for data practitioners to discover them comprehensively. Besides, one analyst complained, *"Sometimes deep hierarchical data really drives me crazy (when seeking to validate data)."* To ease their burden, our system should automatically detect potential quality issues as much as possible.

R4: Depict detected issues. To assist data practitioners in gaining a global knowledge of the data quality, our system should present an overview of quality issues. In addition, to confirm whether the automatically detected issues are reasonable and important, specific information of these issues (e.g., *Which part of the data is involved in the issues? What is the proportion of the problematic data?*) should be revealed and visualized.

R5: Support manual revision. Some quality issues, such as *inconsistent value* and *correlated arrays* (see Sec. 4.2 for details), are difficult to detect automatically, as they require human knowledge to evaluate. Moreover, the results of automatic detection may not fit the needs of data practitioners. Thus, our system should allow data practitioners to identify, change, and discard quality issues.

R6: Recommend transformations to fix issues. To lower the barrier of fixing issues, our system should recommend appropriate transformations and allow users to configure necessary parameters for

	Consistency	Uniqueness	Conciseness	Completeness	Accuracy	Schema-Rigor
Key	inconsistent key	duplicate key	redundant key	missing key		hierarchical key
Value	inconsistent value inconsistent padding inconsistent data type	duplicate value of an array attributes with the same value	redundant padding	missing value empty value	distribution outlier semantic error incorrect data type	confusing data
K-V Pair		partial duplicate k-v pairs all duplicate k-v pairs	redundant k-v pair		misplaced data	correlated arrays data association
Structure	inconsistent structure		redundant interior structure redundant exterior structure			undesirable structure

Fig. 4: Our taxonomy contains 27 types of quality issues of JSON data under two axes, i.e., data objects and data quality dimensions. Some intersections between the two axes are empty, meaning that we have not observed issues under the corresponding data objects and dimensions. The issues marked with green, orange, and black color represent that they can be exactly, roughly, and hardly detected by rules in JsonCurer, respectively. Detailed data features and examples for these issues and the transformations to fix them can be found in the supplementary material.

each transformation. Furthermore, data practitioners expect to observe the results of transformation on the original data.

R7: **Enable iterative data quality management.** To help data practitioners iteratively improve quality, our system should support continuous quality monitoring, which means that once data changes, the system needs to update the JSON schema, re-detect quality issues, etc. Besides, a transformation history is necessary for data practitioners to recall or document what transformations were utilized alongside the entire quality management process.

4 A Taxonomy of JSON Quality Issues

Kim et al. [56] pointed out that, due to *"a lack of appreciation of the types and extent of dirty data"*, most enterprises pay inadequate attention to data quality. Hence, to improve the quality of JSON data, it is essential to study what quality issues may reside in data and how to address them [62]. To this end, we construct a taxonomy of quality issues for JSON data and summarize their corresponding transformations in Sec. 5.4 to guide the development of JsonCurer.

4.1 Methodology

Although we have interviewed three data practitioners and obtained valuable results, they may not be comprehensive. To enhance the coverage of different scenarios, we collected a dataset containing real-world JSON data with quality issues and potential solutions from various websites, including Kaggle [10], GitHub [5], StackOverflow [13], data.world [3], etc. We limited the data format to JSON and used keywords such as "quality issues" and "problematic/dirty data" to seek target data. We adopted two inclusion criteria during the collection process. First, we focused on the scenarios of data analysis and data quality assurance. Specifically, we picked dirty JSON data with analysis tasks or quality requirements. Second, we selected and retained JSON data that provides solutions for improving quality. These solutions could be in the form of descriptive text or cleansing code. At last, we obtained a dataset containing 119 dirty JSON files.

To identify quality issues, we first conducted a comprehensive literature review [43,44,58,62,76,88] regarding data quality dimensions and issues. Based on the descriptions of these dimensions in the literature, we then analyzed the issues extracted from both the interview feedback and the JSON dataset and classified them into different quality dimensions. The majority of these issues could align with well-established quality dimensions, including consistency, uniqueness, conciseness, completeness, and accuracy. However, we found that issues related to JSON schema were difficult to categorize under existing dimensions. Therefore, we introduced a new dimension, schema-rigor, to describe these issues. Consequently, we derived a total of six data quality dimensions. For implementation purposes, we further subdivided them into different data objects they involved (as described in Sec. 3.1). Thus, we constructed a taxonomy of quality issues based on six dimensions and four data objects. Thereafter, we summarized the data feature for each issue in this taxonomy. For example, the data feature of *inconsistent key* issue is that there are different *keys* referencing the same object (such as "genres" and "genre" in Fig. 2). Last, we combined the quality issues with the same data feature into one issue type and created code to label it. Eventually, we acquired a taxonomy containing 27 types of quality issues in total (as shown in Fig. 4). We acknowledge that these issues are not exhaustive, as there could theoretically be additional issues that we are currently unaware of. However, they could give an impression of the research conducted in the area of data quality management.

4.2 Data Quality Dimensions

In this subsection, we describe the six dimensions and their quality issues in our taxonomy.

Consistency. *Inconsistent key/value* means there are different keys/values in *dicts* of an *array* that represent the same entity, which is similar to the entity resolution problem in databases [70]. *Inconsistent padding/data type* means the padding format/data type of values in the same key is inconsistent. *Inconsistent structure* means data with the same structure is arranged and presented by different structures.

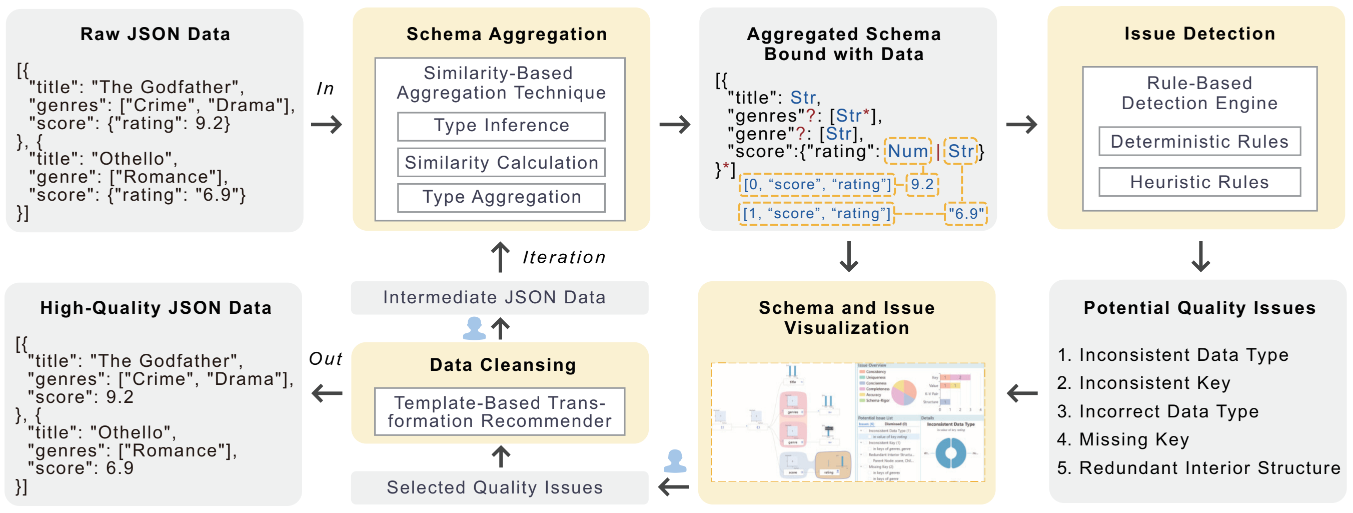
Uniqueness assesses whether the same data object appears multiple times. *Duplicate key* means there is more than one key with the same name in a *dict*. It is valid in JSON syntax while not recommended as it causes ambiguity when accessing the key's value. *Attributes with the same value* means all keys in a *dict* have the same value. *Partial/All duplicate k-v pairs* means the same key in partial/all *dicts* of an *array* has the same value. We distinguish *all* from *partial* because both of the two issues appear frequently in our collected dataset and the transformations to fix them are diverse.

Conciseness emphasizes representing data in a concise and efficient way, avoiding redundancy. *Redundant key/k-v pair* means there are multiple semantically identical keys with fully/semantically identical values in a *dict*. *Redundant padding* means a value has redundant padding, such as unnecessary spaces at both ends of a *string*. *Redundant interior structure* means a structure has redundant data objects across hierarchy levels. *Redundant exterior structure* means there are different structures in a *dict*/*array* that express the same amount of information.

Completeness. *Missing key* means some *dicts* in an *array* have a specific key while the rest do not. *Missing value* means a value is not recorded (i.e., *null*), while *empty value* represents a value that is recorded but conveys nothing (such as an empty *string* or *array*).

Accuracy refers to unexpected values, formats, distributions, or associations [76]. *Distribution outlier* means some values of the same key in *dicts* of an *array* differ significantly from other values. *Semantic error* means some values do not conform to semantics or conditions (such as a negative value in age). *Incorrect data type* means values should have been one data type but are incorrectly represented by another. *Misplaced data* means the values of several keys in a *dict* are misplaced, i.e., the value of a key should belong to another key.

Schema-rigor refers to whether the schema of data is reasonably designed for accessibility, including the suitability of representation [58]. *Hierarchical key* means hierarchy information is embedded in the names of keys. Those names usually have a common prefix. *Confusing data* means the values of an *array* have different semantics, which should be constructed to a new *dict* to represent the semantics clearly. *Correlated arrays* means there are at least two keys in a *dict* whose value types are *arrays* and their corresponding elements are associated semantically. *Data association* means the values of keys in a *dict* should belong



```
[{
  "title": "The Godfather",
  "genres": ["Crime", "Drama"],
  "score": {"rating": 9.2}
}, {
  "title": "Othello",
  "genre": ["Romance"],
  "score": {"rating": "6.9"}
}]
```

```
[{
 "title": Str,
 "genres"?: [Str*],
 "genre"?: [Str],
 "score":{"rating": Num | Str}
}*]
 [0, "score", "rating"] 9.2
 [1, "score", "rating"] "6.9"
```

```
[{
  "title": "The Godfather",
  "genres": ["Crime", "Drama"],
  "score": 9.2
}, {
  "title": "Othello",
  "genres": ["Romance"],
  "score": 6.9
}]
```

Fig. 5: JsonCurer follows a systematic data workflow to enhance the quality of JSON data. The process begins with the input of raw JSON data, followed by four iterative major procedures (highlighted with yellow background), and ultimately outputs cured JSON data with high quality.

to a new attribute, and the names of these keys themselves should be regarded as the values of another attribute. The data with this schema issue is not easily maintainable or understandable. *Undesirable structure* means the JSON structure is not under the desired format.

5 JsonCurer Workflow and Techniques

5.1 System Workflow

We implement JsonCurer as a client-server web-based application using Flask on the server side and Vue and D3 [26] on the client side. Figure 5 illustrates the overall workflow of JsonCurer, comprising four key procedures, i.e., schema aggregation, issue detection, schema and issue visualization, and data cleansing. Specifically, when raw JSON data is input into JsonCurer, the server first extracts and generates an aggregated schema bound with data (R1) through a similarity-based aggregation technique (Sec. 5.2). Based on the aggregated schema, the server then utilizes a rule-based detection engine (Sec. 5.3) to automatically detect potential quality issues (R3). Next, the client-side interface depicts the schema and detected issues via a hierarchical tree rendered with summary visualizations and Bubble Sets (R1, R2, R4). Users can assess the quality of JSON data by inspecting these visualizations and adjust quality issues as needed through interactions (R5). Once users select some issues to address, a template-based transformation recommender (Sec. 5.4) will recommend appropriate transformations with corresponding parameters for configuration (R6). After that, the server performs the user-specified transformation and outputs intermediate JSON data. The whole process of improving quality is iterative and continues until satisfactory high-quality data is obtained (R7).

5.2 Similarity-Based Aggregation Technique

The plain JSON schema presented in Fig. 3B is limited in scalability, as it is inefficient in helping grasp data and identify issues when dealing with large amounts of JSON data. Taking the MongoDB collection in Fig. 2 as an example, if there are hundreds or even thousands of documents in the collection, it becomes challenging for users to inspect the consistency of keys in each document. Therefore, it is necessary to aggregate the common parts in the schema to assist users in comprehending data and discovering its quality issues more conveniently. To this end, we propose a similarity-based aggregation technique to overcome the scalability issue of the plain JSON schema. This technique processes raw JSON data as input, and after three steps, i.e., type inference, similarity calculation, and type aggregation, it outputs an aggregated schema bound with data.

5.2.1 Type Inference

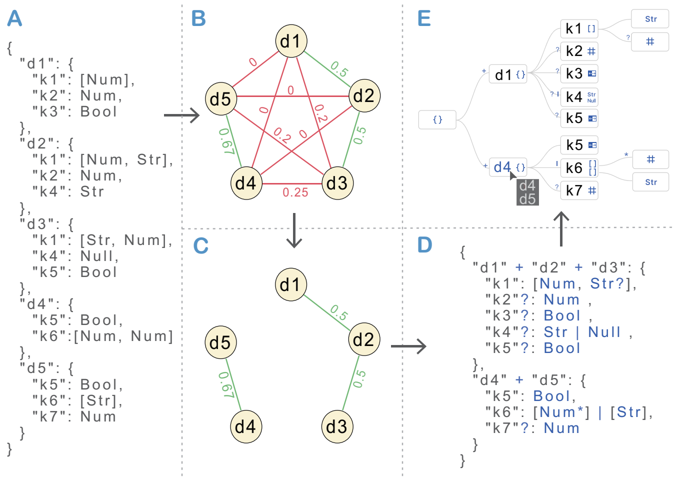
To unravel the hierarchies of the input JSON data, this step drills down into the nested structures (i.e., complex type values, namely, *dicts* and *arrays*) and infers the data type for each simple type value, resulting in a plain JSON schema (e.g., Fig. 3B) for subsequent aggregation.

5.2.2 Similarity Calculation

Given a plain JSON schema, this step calculates the structure similarity within and between complex type values level by level to judge whether these values can be aggregated. In this work, we consider two *dicts* (or *arrays*) can be aggregated if their structures are similar. We adopt the Jaccard index [29, 45] to compute their structure similarity:

$$S(v_i, v_j) = \frac{|StrucSet(v_i) \cap StrucSet(v_j)|}{|StrucSet(v_i) \cup StrucSet(v_j)|} \tag{1}$$

where v_i represents a *dict* (or *array*) value and $StrucSet(v_i)$ returns the set of its keys (or data types of its elements). Note that, $0 \leq S \leq 1$. The larger S indicates that the structures of the two values are more similar. If the similarity is greater than or equal to the predefined threshold (we initialize it to 0.5 and allow users to modify it interactively), we aggregate these values. For instance, the structure similarity of the two documents in the collection (Fig. 2) is $2/4 = 0.5$. Thus, they can be aggregated together.



```
{
  "d1": {
    "k1": [Num],
    "k2": Num,
    "k3": Bool
  },
  "d2": {
    "k1": [Num, Str],
    "k2": Num,
    "k4": Str
  },
  "d3": {
    "k1": [Str, Num],
    "k4": Null,
    "k5": Bool
  },
  "d4": {
    "k5": Bool,
    "k6":[Num, Num]
  },
  "d5": {
    "k5": Bool,
    "k6": [Str],
    "k7": Num
  }
}
```

```
{
  "d1" + "d2" + "d3": {
    "k1": [Num, Str?],
    "k2"?: Num ,
    "k3"?: Bool ,
    "k4"?: Str | Null ,
    "k5"?: Bool
  },
  "d4" + "d5": {
    "k5": Bool,
    "k6": [Num*] | [Str],
    "k7"?: Num
  }
}
```

Fig. 6: The process of generating an aggregated schema tree. Given a plain JSON schema (A), JsonCurer first calculates the similarity of each two values to obtain a complete graph (B), then removes edges less than the threshold (0.5), resulting in connected subgraphs (C), next aggregates the values in each subgraph, yielding an aggregated JSON schema (D), which is finally visualized as a schema tree (E).

In real-world cases, it is quite common to contain more than two complex type values in a *dict*/*array*. Therefore, we further employ the concept of connected components in an undirected graph to determine which of these values can be aggregated. Taking the plain JSON schema in Fig. 6A as an example, there are five key-value pairs in the root *dict*, and all these values are of *dict* type. To aggregate within the root *dict*, we first calculate the similarity of each two *dicts*, resulting in a complete weighted graph (Fig. 6B) where each node represents a *dict* and the weight of each edge denotes the similarity of its corresponding *dicts*. Then, we remove edges below the threshold (0.5), and yield two connected subgraphs (Fig. 6C), which will be aggregated separately.

5.2.3 Type Aggregation

After calculating the similarity, we leverage four rules to elaborate on how to aggregate complex type values in different modes (i.e., within and between). In this work, we refer to a simple but expressive JSON type language proposed by Baazizi et al. [22] which uses regular expressions to describe JSON schema. However, this language does not support the *Within Dict* aggregation scenario, we extend it to adapt to our goal. The four rules are:

Between Arrays: When aggregating between multiple similar *arrays*, each type in these *arrays* will be retained only once. If a certain type does not appear in an *array*, the type will be decorated with a question mark (?) to indicate that it can be optional. For example, the two *arrays*, `[Str, [Num]]` and `[Str, [Str]]`, will be merged into `[Str, [Num]?, [Str]?]`. Note that though the two *subarrays*, `[Num]` and `[Str]`, are of *array* type, they are considered as two different structure types as their similarity is 0.

Within Array: When aggregating within an *array*, multiple identical types in this *array* will be collapsed and marked with an asterisk (*) to represent repetition. For example, the *array* `[[Str], [Num, Num], [Str], Null]` will be transformed into `[[Str]*, [Num*], Null]`.

Between Dicts: When aggregating between multiple similar *dicts*, each key in these *dicts* will be retained only once. If a certain key does not appear in a *dict*, the key will be decorated with a question mark (?) as well. Besides, the value types of each key in these *dicts* will be recursively aggregated. If a key has multiple data types after aggregation, these data types will be separated by a vertical bar (|) to indicate that the value type has different alternatives. For example, the last two *dicts* (i.e., "d4" and "d5") in Fig. 6A can be merged into the last *dict* (i.e., "d4" + "d5") in Fig. 6D.

Within Dict: When aggregating within a *dict*, its child values of *dict* type whose structures are similar will be fused (see *Between Dicts* for aggregation). The corresponding keys of these values will be united with a plus sign (+), denoting that the values of these keys share a similar structure. For example, the root *dict* in Fig. 6A will be merged into the aggregated *dict* in Fig. 6D.

In the process of type aggregation, we also gather the data values and their hierarchy paths for each type (as shown in Fig. 5) for data profiling and the detection and resolution of quality issues. Finally, this step outputs an aggregated JSON schema associated with data.

5.3 Rule-Based Detection Engine

To automatically detect potential issues in JSON data (R3), we first divide the 27 issues in our taxonomy into three classes (see Fig. 4) based on their data features. The first class of issues (12 in total) are deterministic. For example, to identify the *inconsistent data type* issue, we can check whether values are of multiple data types. The second class of issues (6 in total) can be roughly detected by heuristic methods. For instance, human knowledge is needed to determine whether there are *inconsistent keys* in an *array* of *dicts*. But we find that, usually, each *dict* in this *array* has only one of these *inconsistent keys* and the cumulative proportion of these keys appearing in the *array* is 1. Hence, we roughly determine the keys that meet these features as *inconsistent keys*. However, we cannot guarantee these rules are suitable for all situations. The third class of issues (a total of 9) necessitate domain knowledge for identification and cannot be detected automatically, such as *misplaced data*. Then, we develop an issue detection engine that traverses the substructures of the aggregated schema and the data bound to each type to uncover the first two classes of issues. Finally, the engine outputs a set of potential quality issues where each records the data and hierarchy paths it involves.

5.4 Template-Based Transformation Recommender

To help users fix issues, we develop a template-based transformation recommender to recommend proper transformations with parameters for users to configure (R6). To this end, we further analyze the solutions to the 27 quality issues obtained from the interview feedback and our collected dataset (see Sec. 4.1) through the following three steps. First, we abstract each solution to these issues into a transformation template. These templates contain several parameters so that they can be applied to other data or task requirements. Second, we examine the applicability and generalizability of these templates. Third, we pick proper templates by removing those that are low applicability and generalizability or can inevitably introduce additional issues. Our final transformation templates for each issue can be found in the supplementary material. Note that some issues may have no template after examination and selection. After a user has chosen one transformation and configured its parameters, JsonCurer will transform the JSON data through the hierarchy paths bound to the aggregated schema.

6 JSONCURER INTERFACE

The visual interface of JsonCurer encompasses four views (Fig. 1): A) The Data View presents the raw/intermediate JSON data and highlights data differences when performing a specific transformation (R6). B) The Schema View displays an aggregated schema tree (R1) rendered with summary visualizations (R2) and Bubble Sets (R4) to facilitate comprehending data and identifying quality issues. C) The Data Quality Management View provides the overview and details of potential issues (R4) and recommends suitable transformations to fix them (R6). D) The Cleansing History View chronicles all performed transformations (R7) and supports undo operations to allow data rollback to its previous states. Particularly, the Schema View and the Data Quality Management View play crucial roles in helping discover and fix issues. In this section, we describe the visual design of the two views in detail.

6.1 Schema View

After users upload or copy-paste raw JSON data in the Data View, JsonCurer will generate an aggregated schema bound with data and detect potential quality issues in sequence. The Schema View aims to visualize these results in a hierarchical structure.

Schema representation. To visualize the hierarchical aggregated schema (R1), the root of the schema, each k-v pair in a *dict*, and each element in an *array* will be rendered as a tree node, and their hierarchies are represented by edges. As depicted in Fig. 6E, each tree node is rectangular and contains at least one icon representing its data type. We identify k-v pair nodes with the key names. Besides, if the values of a node are of multiple data types (e.g., **"k4"** and **"k6"**), type icons are located vertically in the right of the node. And each icon can connect to their child nodes if they are complex types (e.g., **"k6"**). Additionally, we place the four operators of regular expressions (i.e., **?**, *****, **|**, and **+** mentioned in Sec. 5.2.3) on the left side of nodes to show detailed information about the aggregated schema.

Summary visualization. JsonCurer supports data profiling to help users understand data and discover issues (R2), especially the third class of issues in Fig. 4 (e.g., *redundant key* and *semantic error*). To this end, we first investigate literature [38, 51, 57, 69, 74, 94] discussing appropriate chart types for visualizing different data types or tasks. With the goal of discovering issues, we summarize some visualizations for each data type. Details are discussed in the appendix in supplementary material. Then, we plot the summary visualizations for the data bound to each data type above its node (see Fig. 1B).

Issue presentation. Each quality issue involves a set of nodes in the schema tree. Visualizing these issues (R4) is akin to the problem of presenting node sets along a hierarchical structure. Various techniques have been proposed for visualizing sets. Some of them rely on specific layouts such as matrix [60], radial [19], and treemap [20], which are not suitable for representing sets within a tree. Others can overlay set

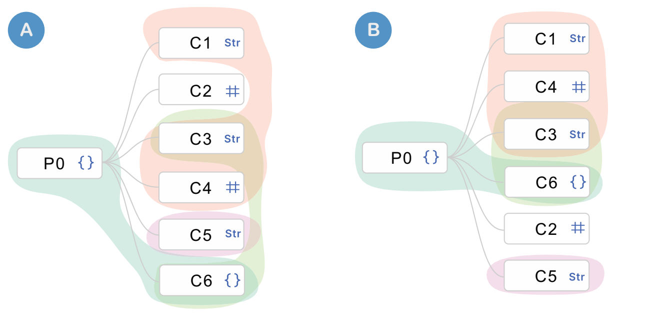


Fig. 7: An example of reordering nodes from A to B based on our algorithm, which is described in detail in the supplementary material.

members on existing visual elements, making them applicable to a tree context. However, some of these overlay techniques [18, 39, 63] are line-based, which may cause confusion when lines intersect with the links in the tree and reduce readability due to line crossings. In contrast, region-based overlay techniques [28, 35, 47] surround set elements with smooth contours, which can clearly express set relations while maintaining the readability of the tree structure. Among them, we selected an open-source implementation of Bubble Sets [35] to visualize issues in JsonCurer. As depicted in Fig. 7, each bubble represents a potential issue, and its background color is encoded to a quality dimension. Besides, it encloses at least one node indicating that the issue involves the data in these nodes. We notice that nodes in some bubbles may be too far apart, causing the intersection of bubbles and separation of focuses (see Fig. 7A). To alleviate this problem, we employ a node reordering algorithm that aims to minimize the vertical distance between the nodes of each bubble (see Fig. 7B).

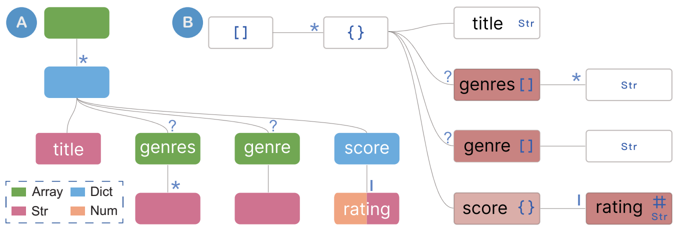


Fig. 8: Design alternatives for visualizing the aggregated schema (A) and the detected quality issues (B) in a hierarchical tree.

Design alternatives. As a hierarchical structure, JSON data is typically represented using node-link trees [48, 90]. Based on the tree representation, we have explored different encoding strategies during the iterative development process of JsonCurer. One candidate for schema representation (Fig. 8A) adopts a vertical hierarchy layout with nodes arranged horizontally, utilizing colors to encode the data types of nodes. However, this node arrangement layout does not align effectively with the original arrangement of JSON data. Moreover, color encoding for data types is less intuitive compared to using icons with semantic meaning. Fig. 8B shows an alternative method that employs a heatmap to convey the severity of quality issues within the tree. While this method effectively communicates the extent of issues associated with individual nodes, it fails to express the set relations of issues involving multiple nodes. Additionally, it lacks the ability to reveal which quality dimension each issue falls under.

Interactions. Users can pan, zoom, fold, and unfold the schema tree and inspect the summary visualizations to understand the JSON data. After a user clicks a summary visualization, the Data View will locate and highlight the corresponding portion (Fig. 1A1). A user can select some nodes in the schema tree and right-click them to identify a new quality issue (Fig. 1B2) so that JsonCurer can recommend transformations to fix it. Besides, in the upper right corner of the Schema View, a user can modify the *dict* and *array* similarity thresholds to reaggregate the JSON schema and click the switch button to toggle the display of summary visualizations above nodes.

6.2 Data Quality Management View

The Data Quality Management View includes four panels to help users inspect and fix issues. Specifically, to present an overview of quality issues (R4), the **issue overview** panel (Fig. 1C1) utilizes a pie chart and a stacked bar chart to visualize the frequency distributions of issues across different quality dimensions and data objects, i.e., the two axes of our taxonomy, respectively. The **potential issue list** panel (Fig. 1C2) exhibits all detected issues grouped by issue type and identified based on their position within the schema tree. This identification enables users to pinpoint where each issue manifests in the data. Users can click one issue to inspect its detailed information (R4) in the **details** panel (Fig. 1C3), which uses a pie chart to display the proportion of problematic data, followed by summary visualizations to help users determine whether the issue needs to be fixed. After inspection, users can disregard less critical issues or false positives by moving them to the Dismissed list (R5). Users can also select some issues to fix in the **data cleansing** panel (Fig. 1C4), which presents recommended transformations with parameters for users to configure (R6). The Preview button is used to examine the impact of a configured transformation on data (R6), as shown in Fig. 1A2. After examination, users can click the Confirm button to transform the JSON data, and the operation will be logged in the Cleansing History View (R7).

Interactions. Users can click the legend in the issue overview panel to select specific quality dimensions for investigation. Issues that are not under these dimensions will be removed from the potential issue list panel, and their corresponding bubbles in the schema tree will also disappear. Besides, clicking a bubble will display the details of the corresponding issue in the details panel. Similarly, clicking an issue in the potential issue list will navigate its involving nodes of the corresponding bubble.

7 Evaluation

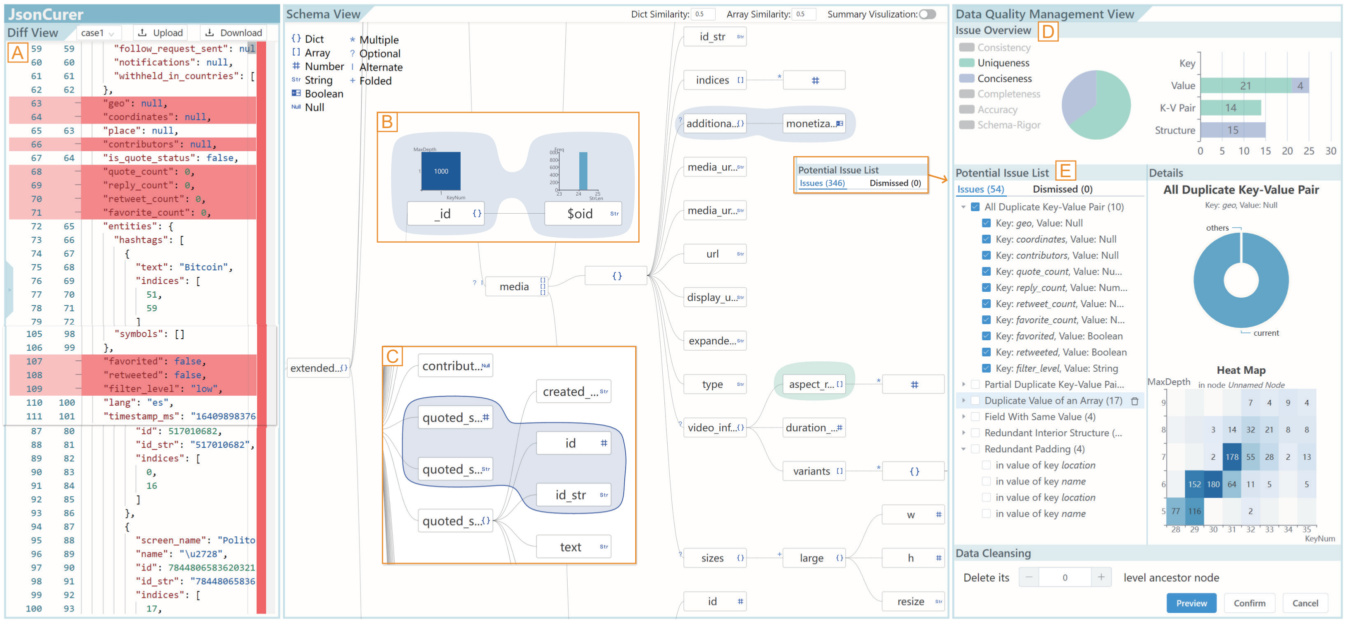
To evaluate the effectiveness and usability of JsonCurer, we conducted two case studies with five data practitioners (P1-P5), where P1-P3 are data analysts and P4-P5 are data stewards. P1, P2, and P4 are the three practitioners we collaborated with in our requirement analysis (Sec. 3.2). P3 and P5 have an average of 3 years of experience in dealing with quality issues of JSON data and are interested in using JsonCurer to improve data quality. The two case studies demonstrate that JsonCurer is applicable to different scales of JSON data under two scenarios, i.e., reliable data analysis and quality assurance in MongoDB.

7.1 Data and Procedure

We prepared two JSON files for evaluation, each corresponding to one case study. The JSON file for case 1 was derived from real undergraduate information (e.g., ‘birthdate’, ‘club’, etc.) obtained from a college class. To reveal the capability of JsonCurer in discovering and fixing issues, we carefully reconstructed it to increase the diversity of issues. This file has a size of 0.16 MB, containing 52 student records with a hierarchy depth of 5, and consists of 20 nodes in the aggregated schema. For case 2, we used an excerpt of a MongoDB dataset [1], including raw bitcoin tweets that contain numerous duplicate and redundant data that required cleansing. The file has a size of 6.73 MB, containing 1000 tweet records with a hierarchy depth of 10, and consists of 514 nodes in the aggregated schema. The case studies were conducted in a think-aloud protocol, and we took notes about their thoughts and comments for further analysis. Each study began with a training session (17 minutes on average) to ensure that the practitioners were familiar with our system. Thereafter, participants freely utilized JsonCurer to improve the quality of the JSON file (15 minutes). Finally, we conducted a semi-structured interview to collect their feedback (18 minutes). Overall, each study took approximately 50 minutes.

7.2 Improving Data Quality for Reliable Analysis

This case demonstrates how JsonCurer helps analysts improve the quality of a small-scale JSON file for reliable analysis. In this case study, the analysts (P1-P3) were asked to independently explore whether the



```
          "follow_request_sent": nul
          "notifications": null,
          "withheld_in_countries": [
        },
        "geo": null,
        "coordinates": null,
        "place": null,
        "contributors": null,
        "is_quote_status": false,
        "quote_count": 0,
        "reply_count": 0,
        "retweet_count": 0,
        "favorite_count": 0,
        "entities": {
          "hashtags": [
            {
              "text": "Bitcoin",
              "indices": [
                51,
                59
              ]
          "symbols": []
        },
        "favorited": false,
        "retweeted": false,
        "filter_level": "low",
        "lang": "es",
        "timestamp_ms": "16409898376
              "id": 517010682,
              "id_str": "517010682",
              "indices": [
                0,
                16
              ]
            },
            {
              "screen_name": "Polito
              "name": "\u2728",
              "id": 784480658362032l
              "id_str": "78448065836
              "indices": [
                17,
```

Fig. 9: System interface of the second case. Data stewards aim to keep data unique and concise (D). Issues under the two dimensions are listed in (E). (A) presents a diff view of employing a batch transformation to delete duplicate keys. (B) and (C) are two examples of redundant data.

age of students and the number of clubs they participate in would affect their course scores. To obtain reliable results, they needed to first discover and fix quality issues in the data that might hinder the analysis.

After uploading the JSON file into JsonCurer, the analysts sought to have a forest view of the data. By glancing at the schema tree (Fig. 1B), they noticed that each student's information was structured as a *dict* with 7 child nodes. Then, the analysts aimed to identify the target nodes related to the analysis task, obtaining three groups of nodes, i.e., ('club') for the number of clubs, ('birth_year', 'birth_month', 'birth_day') for the age, and ('scores', 'secScore') for the course scores. All these nodes were enclosed in bubbles, indicating potential quality issues. To identify whether these issues would affect the analysis results, they next clicked each bubble to scrutinize the details of the issue.

For the 1st group, two issues were detected in the 'club', i.e., *empty value* and *inconsistent data type*. Specifically, the analysts found the 'club' values were of three kinds: an *array* of *strings*, an empty *array*, and a *string* (Fig. 1B1). The value of 'club' being an empty *array* is reasonable when a student did not participate in any clubs. Thus, they dismissed the *empty value* issue. To verify the *inconsistent data type* issue, they reviewed the summary visualizations above the 'club' node and clicked them to locate data in the Data View (Fig. 1A1), finding that if students only participated in one club, its value was the club name itself. To fix this issue, they configured parameters of a recommended transformation that splits a *string* into an *array* in the data cleansing panel. After previewing the transformation result (Fig. 1A2), they clicked the Confirm button to employ the transformation.

For the 2nd group, a *hierarchical key* issue was detected on the three nodes. To calculate the student's age, the analysts intended to merge these nodes into a single *string* so that it could be converted to a date. After inspecting the detailed information of the issue (Fig. 1C3), they discovered that there was a 'birth_month' whose string length was 2 while the others were 3. From the category frequency bar chart of the 'birth_month' node, they found that some string was shown as 'Fe', which might be an incorrect abbreviation for February. To generate the birthdate correctly, they selected and right-clicked the 'birth_month' node and manually identified a new issue, i.e., *inconsistent value*, to the node (Fig. 1B2). Thereafter, they fixed the issue by replacing 'Fe' with 'Feb'. At last, they created a new node 'birthdate' by concatenating the three 'birth' nodes to fix the *hierarchical key* issue.

For the 3rd group, the analysts found that the 'secScore' node involved a *missing key* issue and the child node of 'scores' involved a *missing value* issue. From the tree structure, they noticed that the two nodes were the keys of several major courses (marked with **+**), which were aggregated together by JsonCurer. To figure out which major courses were missing 'secScore', they increased the *dict* similarity threshold to 1. After regenerating the schema, they found the two issues stemmed from the major course 'Experiments and Practice' (Fig. 1B3). Therefore, they deleted the course to fix the two issues. Finally, the analysts exported the cured data for the downstream analysis task.

7.3 Deduplicating MongoDB Data for Quality Assurance

In this case, the stewards (P4-P5) were required to eliminate duplicate and redundant content in the second JSON file as much as possible. The deduplication task is a common requirement for data stewards to maintain data quality in databases. This case demonstrates how data stewards can improve their efficiency in deduplicating large-scale data in MongoDB with the aid of JsonCurer.

After uploading the tweet JSON file, JsonCurer completed the loading process in 15 seconds, during which it successfully detected a total of 346 quality issues. To deduplicate data, the stewards kept the issues under the uniqueness and conciseness dimensions by deselecting the other dimensions in the issue overview (Fig. 9D), resulting in 54 issues remaining, which fell into six issue types where four were duplicate and two were redundancy, as shown in Fig. 9E.

The stewards first focused on the four types of duplicate issues. They observed that there were ten keys whose values were all the same (*all duplicate k-v pair*) and four keys with partially duplicate values (*partial duplicate k-v pair*). After inspecting their corresponding summary visualizations, the stewards presumed that the values of the ten keys (such as *null* and 0) did not provide valuable information. Thus, they selected all the ten issues and employed a batch transformation to delete these keys simultaneously (Fig. 9A). As for the four *partial duplicate k-v pair* issues, they believed that these issues were reasonable and discarded them together. Thereafter, the stewards checked and fixed the remaining duplicate issues using the same way described above.

Then the stewards moved to the two types of redundancy issues. They found there were several *redundant interior structures*. An example was shown in Fig. 9B, the key '_id' had only one child key '$oid' and the two keys represented the same meaning. However, such 'redundant' structure was reasonable as it was automatically generated by MongoDB to ensure the uniqueness of each document in the collection. Hence, the stewards decided to ignore it and flatten the other redundant nested structures. Next, they reviewed the *redundant padding* issues and found some values of these keys had extra whitespace at the end.

To fix these issues, the stewards also employed a batch transformation (i.e., trim) to remove the whitespace.

After that, all the 54 detected issues had been scanned. The stewards continued to explore the aggregated schema and discovered redundant contents that JsonCurer did not detect (see Fig. 9C). The values of the four keys denoted the same content, which could be reduced to a single one. And P4 added that how to reduce them depended on the specific business requirements.

7.4 Feedback and Suggestions

All practitioners were deeply impressed by our system and spoke highly of our proposed design, including the aggregated schema visualization and automatic issue detection. As P2 commented, *"it's a useful and intuitive tool that I have never seen before."* P5 appreciated the usability of JsonCurer, *"it can help me quickly discover and remove numerous duplicate and redundant data."* P3 and P4 mentioned that although JsonCurer was somewhat complex (due to the need to understand various issues), once they were familiar with it, it was more efficient than checking and fixing issues by manually writing code. In addition, they also put forward constructive suggestions for improvement.

Schema design: (1) If there were a plethora of nodes in the schema tree, practitioners needed to repeatedly pan and zoom the tree to view the data structure or locate specific nodes of interest. To alleviate the issue, P4 suggested that the Schema View could be equipped with a minimap for the schema tree, while P5 expected to be able to navigate a node by entering its hierarchy path. (2) P1 noted that setting a proper similarity value can be difficult, and the new similarity threshold may affect the aggregation results of the other subtrees. Thus, it is not user-friendly to control the aggregation of multiple subtrees by adjusting the similarity. He recommended that users could directly split and merge nodes through interactions such as clicking and dragging. (3) P3 hoped to directly manipulate the nodes and their visualizations in the Schema View to interactively perform data transformations, such as deleting redundant nodes, rearranging inconsistent structures, etc.

Issue detection and resolution: (1) Most practitioners suggested that JsonCurer should support detecting user-defined issues (e.g., verifying integrity constraints) and employing basic transformations (e.g., *delete*, *rearrange*, and *flatten*) to apply to customized requirements. (2) P5 was confused about which issues to fix first. He suggested assigning a priority for each issue type, with higher priority indicating that the issue was critical and required to fix first. (3) When a new JSON data is generated or a previous one is updated with new entities, it would be tedious to repeatedly improve the quality of the new data with the same cleansing strategies. Thus, both P2 and P4 hoped that JsonCurer could record the data transformation operation as a script, and execute the script to solve the issues for new data.

8 Discussion

Evaluation: The taxonomy of quality issues we constructed has not been systematically assessed. Thus, it is unclear whether the 27 issues can cover the majority of issues that users may encounter, whether the issue detection rules are reasonable, and to what extent our system can help improve users' work efficiency. In the future, we plan to conduct more comprehensive studies with a broader user population. Moreover, to evaluate the efficiency and scalability of JsonCurer in larger datasets, we plan to provide a benchmark on how the performance of JsonCurer gets affected as a function of the data size and the hierarchical depth.

Lessons Learned: We provide two design lessons learned during the development of JsonCurer. First, the feedback from case studies shows that data practitioners were satisfied with the intuitiveness of our visual design. However, we notice that if multiple bubbles with different dimensions intersect, the color of the overlapping area will change, causing the loss of dimension information. Based on the suggestions we received, we plan to use the color opacity of bubble to encode the priority of its corresponding issues. The higher the priority of the issue, the darker the color of the bubble will be. This could guide users to fix quality issues in priority order more intuitively. Second, the use of schema aggregation in JsonCurer has proven to be effective in improving users' understanding of semi-structured data. By organizing data into structured and hierarchically grouped components, users can efficiently manage data quality across the four activities mentioned in Sec. 2. However, due to the schema-less feature of semi-structured data, the similarity-based aggregation technique we employed may lead to inaccurate aggregations. For instance, data groups with different semantic meanings but similar structures might be incorrectly merged, hindering effective data quality management. To address this issue, we plan to explore the use of large language models to improve the accuracy of schema aggregation.

Generalizability: Though JsonCurer focuses on data quality management for JSON data, we believe that our approach is generalizable to other semi-structured data formats, such as XML and YAML, as they can be converted to each other by numerous tools [2, 34]. However, during the conversion process between these formats, one potential issue that may arise is data loss or corruption [27, 75], such as data type mismatch and namespace loss when converting from XML to JSON. To mitigate the risk of data loss, data mapping and normalization techniques need to be considered to ensure all relevant data is preserved throughout the conversion process.

9 Conclusion

In this work, we present JsonCurer, an interactive visualization system for improving the quality of JSON data. JsonCurer can automatically detect quality issues and recommend proper transformations to fix them. To guide the detection and recommendation, we construct a taxonomy of issues based on interviews with data practitioners and a review of our collected dataset. We propose a similarity-based technique to aggregate JSON schema and visualize it as a tree to display structural information. In this tree, summary visualizations are given above each node to present statistical features, and Bubble Sets across nodes are used to illustrate quality issues. Two case studies show that JsonCurer is effective and helpful in improving data quality. In future work, we aspire to enhance its scalability and usability, and explore its potential opportunity and research direction, as described below.

Scalability: Our case studies show that JsonCurer is capable of handling data with up to 500 nodes. However, the studies reveal two scalability issues. First, P4 and P5 reported difficulties in navigating nodes of interest within large-scale trees. To alleviate the issue, we plan to create a minimap for the schema tree and support nodes searching by hierarchy paths. Second, as our detection engine is rule-based and customized for quality issues, extending it to a large variety of issues is time-consuming. In future iterations, we intend to optimize JsonCurer by supporting user-defined issues to enhance its usability. In addition, we also plan to allow for more customization from the user side, such as providing configuration or template files that can be shared.

Opportunity: An exciting opportunity for JsonCurer is to improve the quality of JSON datasets used for training machine learning models. High-quality datasets are essential for effective machine learning. Though there have been efforts to enhance the quality of models, little attention has been paid to improving the data quality [49]. We anticipate that JsonCurer can fill this gap by facilitating the discovery and resolution of quality issues (e.g., imbalanced classes and missing/inconsistent labels [93]) in JSON datasets. By utilizing our work, machine learning practitioners and researchers could achieve better model performance.

Research Direction: The transformations provided by JsonCurer are designed for fixing quality issues. However, data practitioners often seek general-purpose transformations to complete specific tasks. For example, in data integration, practitioners usually perform *filter* and *join* operations. However, current prototype of JsonCurer does not support these transformations. Therefore, a potential research direction is to explore what kind of transformations could be utilized by practitioners to wrangle semi-structured data and how to design visualizations and interactions to facilitate the wrangling process.

Acknowledgments

The work was supported by NSFC (U22A2032), National Key R&D Program of China (2022YFE0137800), and Key "Pioneer" R&D Projects of Zhejiang Province (2023C01120). We gratefully thank the anonymous reviewers for their valuable comments.

References

[1] 2022 January Bitcoin Tweets. https://www.kaggle.com/datasets/kodamacodes/2022-january-bitcoin-tweets. Accessed: Jul 2023. 7

[2] Convert JSON to other Formats and Vice-Versa. https://www.convertjson.com/. Accessed: Feb 2023. 9

[3] data.world. https://data.world/. Accessed: Feb 2023. 4

[4] Flatten Complex Nested JSON. https://www.coderstool.com/flatten-json. Accessed: Feb 2023. 3

[5] GitHub. https://github.com/. Accessed: Feb 2023. 4

[6] JSON Crack - Crack your data into pieces. https://jsoncrack.com/. Accessed: Feb 2023. 2

[7] JSON Formatter & Validator. https://jsonformatter.curiousconcept.com/. Accessed: Feb 2023. 3

[8] JSON Formatter, Validator, Viewer, Editor & Beautifier Online. https://www.jsonformatter.io/. Accessed: Feb 2023. 3

[9] JSON To CSV Converter. https://www.convertcsv.com/json-to-csv.htm. Accessed: Feb 2023. 3

[10] Kaggle. https://www.kaggle.com/. Accessed: Feb 2023. 4

[11] OpenRefine. https://openrefine.org. Accessed: Feb 2023. 1, 2, 3

[12] Oracle Enterprise Data Quality. https://www.oracle.com/hk/middleware/technologies/enterprise-data-quality.html. Accessed: Feb 2023. 2

[13] Stackoverflow. https://stackoverflow.com. Accessed: Feb 2023. 4

[14] Tableau Prep Builder. https://www.tableau.com/products/prep. Accessed: Feb 2023. 1

[15] Trifacta. https://www.trifacta.com/. Accessed: Feb 2023. 1, 3

[16] Z. Abedjan, L. Golab, F. Naumann, and T. Papenbrock. Data Profiling. *Synthesis Lectures on Data Management*, 10(4):1–154, 2018. 2

[17] O. Akbulut, L. McLaughlin, T. Xin, M. Forshaw, and N. S. Holliman. Visualizing ordered bivariate data on node-link diagrams. *Visual Informatics*, 7(3):22–36, 2023. doi: 10.1016/j.visinf.2023.06.003 2

[18] B. Alper, N. Riche, G. Ramos, and M. Czerwinski. Design Study of LineSets, a Novel Set Visualization Technique. *IEEE TVCG*, 17(12):2259–2267, 2011. doi: 10.1109/TVCG.2011.186 7

[19] B. Alsallakh, W. Aigner, S. Miksch, and H. Hauser. Radial Sets: Interactive Visual Analysis of Large Overlapping Sets. *IEEE TVCG*, 19(12):2496–2505, 2013. doi: 10.1109/TVCG.2013.184 6

[20] B. Alsallakh and L. Ren. PowerSet: A Comprehensive Visualization of Set Intersections. *IEEE TVCG*, 23(1):361–370, 2016. doi: 10.1109/TVCG.2016.2598496 6

[21] C. Arbesser, F. Spechtenhauser, T. Mühlbacher, and H. Piringer. Visplause: Visual Data Quality Assessment of Many Time Series Using Plausibility Checks. *IEEE TVCG*, 23(1):641–650, 2017. doi: 10.1109/tvcg.2016.2598592 1

[22] M.-A. Baazizi, H. B. Lahmar, D. Colazzo, G. Ghelli, and C. Sartiani. Schema Inference for Massive JSON Datasets. In *Proc. of Extending Database Technology*, 2017. doi: 10.5441/002/edbt.2017.21 2, 3, 6

[23] C. Batini, F. Cabitza, C. Cappiello, and C. Francalanci. A Comprehensive Data Quality Methodology for Web and Structured Data. *International Journal of Innovative Computing and Applications*, 1(3):205–218, 2008. doi: 10.1109/icdim.2007.369236 2

[24] C. Batini and M. Scannapieca. Data Quality: Concepts, Methodologies and Techniques. *Cham: Springer International Publishing*, 2016. doi: 10.1007/3-540-33173-5 2

[25] J. M. Borovina Josko and J. E. Ferreira. Visualization properties for Data Quality Visual Assessment: An exploratory Case Study. *Information Visualization*, 16(2):93–112, 2017. doi: 10.1177/1473871616629516 2

[26] M. Bostock, V. Ogievetsky, and J. Heer. D^3 Data-Driven Documents. *IEEE TVCG*, 17(12):2301–2309, 2011. doi: 10.1109/TVCG.2011.185 5

[27] J. Boyer, S. Gao, S. Malaika, M. Maximilien, R. Salz, and J. Simeon. Experiences with JSON and XML Transformations. In *W3C Workshop on Data and Services Integration*, 2011. 9

[28] H. Byelas and A. Telea. Visualization of Areas of Interest in Software Architecture Diagrams. In *Proc. of the ACM symposium on Software visualization*, pp. 105–114, 2006. 7

[29] S. Cai, S.-H. Hong, X. Xia, T. Liu, and W. Huang. A machine learning approach for predicting human shortest path task performance. *Visual Informatics*, 6(2):50–61, 2022. doi: 10.1016/j.visinf.2022.04.001 5

[30] R. Chen, D. Weng, Y. Huang, X. Shu, J. Zhou, G. Sun, and Y. Wu. Rigel: Transforming Tabular Data by Declarative Mapping. *IEEE TVCG*, 29(1):128–138, 2023. doi: 10.1109/TVCG.2022.3209385 3

[31] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang. Data Cleaning: Overview and Emerging Challenges. In *Proc. of ACM SIGMOD*, pp. 2201–2206, 2016. doi: 10.1145/2882903.2912574 2

[32] X. Chu, I. F. Ilyas, and P. Papotti. Holistic Data Cleaning: Putting Violations Into Context. In *Proc. of the IEEE International Conference on Data Engineering*, pp. 458–469, 2013. doi: 10.1109/icde.2013.6544847 3

[33] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. In *Proc. of ACM SIGMOD*, pp. 1247–1261, 2015. doi: 10.1145/2723372.2749431 3

[34] Code Beautify. JSON Cleaner Online to clean Messy JSON online. https://codebeautify.org/json-cleaner/. Accessed: Feb 2023. 3, 9

[35] C. Collins, G. Penn, and S. Carpendale. Bubble Sets: Revealing Set Relations with Isocontours over Existing Visualizations. *IEEE TVCG*, 15(6):1009–1016, 2009. doi: 10.1109/TVCG.2009.122 7

[36] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. NADEEF: A Commodity Data Cleaning System. In *Proc. of ACM SIGMOD*, pp. 541–552, 2013. doi: 10.1145/2463676.2465327 3

[37] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*, vol. 479. John Wiley & Sons, 2003. 3

[38] Z. Deng, D. Weng, S. Liu, Y. Tian, M. Xu, and Y. Wu. A survey of urban visual analytics: Advances and future directions. *Computational Visual Media*, 9(1):3–39, 2023. doi: 10.1007/s41095-022-0275-7 6

[39] K. Dinkla, M. J. Van Kreveld, B. Speckmann, and M. A. Westenberg. Kelp Diagrams: Point Set Membership Visualization. *Computer Graphics Forum*, 31(3):875–884, 2012. doi: 10.1111/j.1467-8659.2012.03080.x 7

[40] M. Droettboom et al. Understanding JSON Schema. https://json-schema.org/understanding-json-schema/UnderstandingJSONSchema.pdf. Accessed: Feb 2023. 3

[41] D. Durner, V. Leis, and T. Neumann. JSON Tiles: Fast Analytics on Semi-Structured Data. In *Proc. of the International Conference on Management of Data*, pp. 445–458, 2021. doi: 10.1145/3448016.3452809 2

[42] L. Ehrlinger and W. Wöß. Automated Data Quality Monitoring. In *Proc. of the MIT International Conference on Information Quality*, pp. 15.1–15.9, 2017. 2

[43] L. Ehrlinger and W. Wöß. A Novel Data Quality Metric for Minimality. In *Proc. of Data Quality and Trust in Big Data*, pp. 1–15, 2019. doi: 10.1007/978-3-030-19143-6_1 2, 4

[44] L. Ehrlinger and W. Wöß. A Survey of Data Quality Measurement and Monitoring Tools. *Frontiers in Big Data*, 5:1–30, 2022. doi: 10.3389/fdata.2022.850611 1, 2, 3, 4

[45] S. Fletcher, M. Z. Islam, et al. Comparing sets of patterns with the Jaccard index. *Australasian Journal of Information Systems*, 22:1–17, 2018. 5

[46] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The LLUNATIC Data-Cleaning Framework. *Proc. of the VLDB Endowment*, 6(9):625–636, 2013. doi: 10.14778/2536360.2536363 3

[47] J. Heer and D. Boyd. Vizster: Visualizing online social networks. In *IEEE Symposium on Information Visualization*, pp. 32–39, 2005. doi: 10.1109/INFVIS.2005.1532126 7

[48] J. L. C. Izquierdo and J. Cabot. JSONDiscoverer: Visualizing the schema lurking behind JSON documents. *Knowledge-Based Systems*, 103:52–55, 2016. doi: 10.1016/j.knosys.2016.03.020 2, 7

[49] A. Jain, H. Patel, L. Nagalapatti, N. Gupta, S. Mehta, S. Guttula, S. Mujumdar, S. Afzal, R. Sharma Mittal, and V. Munigala. Overview and Importance of Data Quality for Machine Learning Tasks. In *Proc. of ACM SIGKDD*, pp. 3561–3562, 2020. doi: 10.1145/3394486.3406477 9

[50] S. Judah, M. Selvage, and A. Jain. Magic Quadrant for Data Quality Tools. https://www.gartner.com/en/documents/3522717, 2016. Accessed: Jan 2023. 2

[51] K. Kagkelidis, I. Dimitriadis, and A. Vakali. Lumina: an adaptive, automated and extensible prototype for exploring, enriching and visualizing data. *Journal of Visualization*, 24:631–655, 2021. doi: 10.1007/s12650-020-00718-y 6

[52] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. Van Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck, and P. Buono. Research Directions in Data Wrangling: Visualizations and Transformations for Usable and Credible Data. *Information Visualization*, 10(4):271–288, 2011. doi: 10.1177/1473871611415994 3

[53] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer. Profiler: Integrated Statistical Analysis and Visualization for Data Quality Assessment. In *Proc. of the International Working Conference on Advanced Visual Interfaces*, pp. 547–554, 2012. doi: 10.1145/2254556.2254659 1, 2

[54] S. Kasica, C. Berret, and T. Munzner. Table Scraps: An Actionable Framework for Multi-Table Data Wrangling From An Artifact Study of Computational Journalism. *IEEE TVCG*, 27(2):957–966, 2021. doi: 10.1109/TVCG.2020.3030462 3

[55] Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. BigDansing: A System for Big Data Cleansing. In *Proc. of ACM SIGMOD*, pp. 1215–1230, 2015. doi: 10.1145/2723372.2747646 3

[56] W. Kim. On Three Major Holes in Data Warehousing Today. *Journal of Object Technology*, 1(4):39–47, 2002. doi: 10.5381/jot.2002.1.4.c3 4

[57] X. Kui, N. Liu, Q. Liu, J. Liu, X. Zeng, and C. Zhang. A survey of visual analytics techniques for online education. *Visual Informatics*, 6(4):67–77, 2022. doi: 10.1016/j.visinf.2022.07.004 6

[58] N. Laranjeiro, S. N. Soydemir, and J. Bernardino. A Survey on Data Quality: Classifying Poor Data. In *Proc. of the IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 179–188, 2015. doi: 10.1109/PRDC.2015.41 2, 4

[59] Y. W. Lee, D. M. Strong, B. K. Kahn, and R. Y. Wang. AIMQ: A Methodology for Information Quality Assessment. *Information & Management*, 40(2):133–146, 2002. doi: 10.1016/s0378-7206(02)00043-5 2

[60] A. Lex, N. Gehlenborg, H. Strobelt, R. Vuillemot, and H. Pfister. UpSet: Visualization of Intersecting Sets. *IEEE TVCG*, 20(12):1983–1992, 2014. doi: 10.1109/TVCG.2014.2346248 6

[61] G. Li, R. Li, Z. Wang, H. C. Liu, M. Lu, and G. Wang. HiTailor: Interactive Transformation and Visualization for Hierarchical Tabular Data. *IEEE TVCG*, 29(1):139–148, 2023. doi: 10.1109/TVCG.2022.3209354 3

[62] L. Li, T. Peng, and J. Kennedy. A Rule Based Taxonomy of Dirty Data. *GSTF International Journal on Computing*, 1(2):140–148, 2011. 1, 2, 4

[63] W. Meulemans, N. H. Riche, B. Speckmann, B. Alper, and T. Dwyer. KelpFusion: A Hybrid Set Visualization Technique. *IEEE TVCG*, 19(11):1846–1858, 2013. doi: 10.1109/TVCG.2013.76 7

[64] N. Micic, D. Neagu, F. Campean, and E. H. Zadeh. Towards a Data Quality Framework for Heterogeneous Data. In *Proc. of the IEEE International Conference on Internet of Things and IEEE Green Computing and Communications and IEEE Cyber, Physical and Social Computing and IEEE Smart Data*, pp. 155–162, 2017. doi: 10.5220/0005557001890194 2

[65] S. Moore. How to Create a Business Case for Data Quality Improvement. https://www.gartner.com/smarterwithgartner/how-to-create-a-business-case-for-data-quality-improvement. Accessed: Jan 2023. 1

[66] O. Moseler, L. Kreber, and S. Diehl. The ThreadRadar visualization for debugging concurrent Java programs. *Journal of Visualization*, 25(6):1267–1289, 2022. doi: 10.1007/s12650-022-00843-w 2

[67] T. Munzner. A Nested Model for Visualization Design and Validation. *IEEE TVCG*, 15(6):921–928, 2009. doi: 10.1109/TVCG.2009.111 3

[68] F. Naumann. Data Profiling Revisited. *ACM SIGMOD Record*, 42(4):40–49, 2014. doi: 10.1145/2590989.2590995 2

[69] J. Oetting. Data Visualization 101: How to Choose the Right Chart or Graph for Your Data. https://library.cup.edu.cn/upload_files/article/14_20191204084012.pdf, 2019. 6

[70] G. Papadakis, D. Skoutas, E. Thanos, and T. Palpanas. Blocking and Filtering Techniques for Entity Resolution: A Survey. *ACM Computing Surveys*, 53(2):1–42, 2020. doi: 10.1145/3377455 4

[71] H. Park and J. Widom. CrowdFill: Collecting Structured Data from the Crowd. In *Proc. of ACM SIGMOD*, pp. 577–588, 2014. doi: 10.1145/2588555.2610503 3

[72] E. Rahm and H. H. Do. Data Cleaning: Problems and Current Approaches. *IEEE Database Engineering Bulletin*, 23(4):3–13, 2000. 3

[73] V. Raman and J. M. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. In *Proc. of the International Conference on Very Large Data Bases*, pp. 381–390, 2001. 2, 3

[74] B. Saket, A. Endert, and Ç. Demiralp. Task-Based Effectiveness of Basic Visualizations. *IEEE TVCG*, 25(7):2505–2512, 2018. doi: 10.1109/TVCG.2018.2829750 6

[75] B. Šandrih, D. Tošić, and V. Filipović. Towards Efficient and Unified XML/JSON Conversion-A New Conversion. *IPSI BgD Transactions on Internet Research (TIR) vol*, 13, 2017. 9

[76] C. O. Schmidt, S. Struckmann, C. Enzenbach, A. Reineke, J. Stausberg, S. Damerow, M. Huebner, B. Schmidt, W. Sauerbrei, and A. Richter. Facilitating harmonized data quality assessments. A data quality framework for observational health research data collections with software implementations in R. *BMC Medical Research Methodology*, 21(1):1–15, 2021. doi: 10.1186/s12874-021-01252-7 2, 4

[77] L. Sebastian-Coleman. *Measuring Data Quality for Ongoing Improvement: A Data Quality Assessment Framework*. Newnes, 2012. 2

[78] M. Sedlmair, M. Meyer, and T. Munzner. Design Study Methodology: Reflections from the Trenches and the Stacks. *IEEE TVCG*, 18(12):2431–2440, 2012. doi: 10.1109/TVCG.2012.213 2

[79] G. Simões, H. Galhardas, and L. Gravano. When Speed Has a Price: Fast Information Extraction Using Approximate Algorithms. *Proc. of the VLDB Endowment*, 6(13):1462–1473, 2013. doi: 10.14778/2536258.2536259 3

[80] D. Skoutas and A. Simitsis. Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data. *International Journal on Semantic Web and Information Systems*, 3(4):1–24, 2007. doi: 10.4018/jswis.2007100101 2

[81] W. Spoth, T. Xie, O. Kennedy, Y. Yang, B. Hammerschmidt, Z. H. Liu, and D. Gawlick. SchemaDrill: Interactive Semi-Structured Schema Design. In *Proc. of the Workshop on Human-In-the-Loop Data Analytics*, pp. 1–7, 2018. doi: 10.1145/3209900.3209908 2

[82] Y. Tong, C. C. Cao, C. J. Zhang, Y. Li, and L. Chen. CrowdCleaner: Data Cleaning for Multi-version Data on the Web via Crowdsourcing. In *Proc. of the IEEE International Conference on Data Engineering*, pp. 1182–1185, 2014. doi: 10.1109/icde.2014.6816736 3

[83] L. Tuura, A. Meyer, I. Segoni, and G. Della Ricca. CMS Data Quality Monitoring: Systems and Experiences. *Journal of Physics: Conference Series*, 219(7), 2010. doi: 10.1088/1742-6596/219/7/072020 2

[84] Á. Valencia Parra, Á. J. Varela Vaca, M. T. Gómez López, and P. Ceravolo. CHAMALEON: Framework to improve Data Wrangling with Complex Data. In *Proc. of the International Conference on Information Systems*, pp. 1–17, 2019. 1, 3

[85] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A Sample-and-Clean Framework for Fast and Accurate Query Processing on Dirty Data. In *Proc. of ACM SIGMOD*, pp. 469–480, 2014. doi: 10.1145/2588555.2610505 3

[86] J. Wang and N. Tang. Towards Dependable Data Repairing with Fixing Rules. In *Proc. of ACM SIGMOD*, pp. 457–468, 2014. doi: 10.1145/2588555.2610494 3

[87] R. Y. Wang. A Product Perspective on Total Data Quality Management. *Communications of the ACM*, 41(2):58–65, 1998. doi: 10.1145/269012.269022 2

[88] R. Y. Wang and D. M. Strong. Beyond Accuracy: What Data Quality Means to Data Consumers. *Journal of Management Information Systems*, 12(4):5–33, 1996. doi: 10.1145/3368089.3417045 2, 4

[89] Y. Wang, Z. Zhu, L. Wang, G. Sun, and R. Liang. Visualization and visual analysis of multimedia data in manufacturing: A survey. *Visual Informatics*, 6(4):12–21, 2022. doi: 10.1016/j.visinf.2022.09.001 2

[90] Z. Wang, D. Zhou, and S. Chen. STEED: An Analytical Database System for TrEE-structured Data. *Proc. of the VLDB Endowment*, 10(12):1897–1900, 2017. doi: 10.14778/3137765.3137803 2, 7

[91] K. Xiong, S. Fu, G. Ding, Z. Luo, R. Yu, W. Chen, H. Bao, and Y. Wu. Visualizing the Scripts of Data Wrangling With SOMNUS. *IEEE TVCG*, 29(6):2950–2964, 2023. doi: 10.1109/TVCG.2022.3144975 2

[92] K. Xiong, Z. Luo, S. Fu, Y. Wang, M. Xu, and Y. Wu. Revealing the Semantics of Data Wrangling Scripts With COMANTICS. *IEEE TVCG*, 29(1):117–127, 2023. doi: 10.1109/TVCG.2022.3209470 2

[93] W. Yang, M. Liu, Z. Wang, and S. Liu. Foundation Models Meet Visualizations: Challenges and Opportunities. *Computational Visual Media*, 2024. https://arxiv.org/abs/2310.05771. 9

[94] S. Zhu, G. Sun, Q. Jiang, M. Zha, and R. Liang. A Survey on Automatic Infographics and Visualization Recommendations. *Visual Informatics*, 4(3):24–40, 2020. doi: 10.1016/j.visinf.2020.07.002 6