A Benchmark of JSON-compatible Binary Serialization Specifications

Juan Cruz Viotti*
Department of Computer Science
University of Oxford
Oxford, GB OX1 3QD
juancruz.viotti@kellogg.ox.ac.uk

Mital Kinderkhedia
Department of Computer Science
University of Oxford
Oxford, GB OX1 3QD
mital.kinderkhedia@cs.ox.ac.uk

Abstract

In this paper, we present a comprehensive benchmark of JSON-compatible binary serialization specifications using the SchemaStore open-source test suite collection of over 400 JSON documents matching their respective schemas and representative of their use across industries. We benchmark a set of schema-driven (ASN.1, Apache Avro, Microsoft Bond, Cap'n Proto, FlatBuffers, Protocol Buffers, and Apache Thrift) and schema-less (BSON, CBOR, FlexBuffers, MessagePack, Smile, and UBJSON) JSON-compatible binary serialization specifications. Existing literature on benchmarking JSON-compatible binary serialization specifications demonstrates extensive gaps when it comes to binary serialization specifications coverage, reproducibility and representativity, the role of data compression in binary serialization and the choice and use of obsolete versions of binary serialization specifications. We believe our work is the first of its kind to introduce a tiered taxonomy for JSON documents consisting of 36 categories classified as Tier 1, Tier 2 and Tier 3 as a common basis to class JSON documents based on their size, type of content, characteristics of their structure and redundancy criteria. We built and published a free-to-use online tool to automatically categorize JSON documents according to our taxonomy that generates related summary statistics. In the interest of fairness and transparency, we adhere to reproducible software development standards and publicly host the benchmark software and results on GitHub. Our findings provide a number of conclusions: sequential binary serialization specifications are typically more space-efficient than pointer-based binary serialization specifications independently of whether they are schema-less or schema-driven; in comparison to compressed JSON, both compressed and uncompressed schema-less binary serialization specifications result in negative median and average size reductions. Through our analysis, we find that both compressed and uncompressed schema-driven binary serialization specifications result in positive median and average reduction. Furthermore, compressed sequential schema-driven binary serialization specifications are strictly superior to compressed JSON in all the cases from the input data.

1 Introduction

[71] discusses the relevance of the JSON [17] textual schema-less serialization specification in the context of web services, its history, characteristics, advantages and disadvantages of 13 JSON-compatible schema-driven and schema-less binary serialization specifications: ASN.1 [56], Apache Avro [27],

*https://www.jviotti.com

Preprint. Under review.

Microsoft Bond [43], BSON [42], Cap'n Proto [69], CBOR [5], FlatBuffers [66], FlexBuffers [67], MessagePack [29], Protocol Buffers [31], Smile [55], Apache Thrift [60], and UBJSON [7].

One of the conclusions found in [71] is that JSON [17] is neither considered a runtime-efficient nor a space-efficient serialization specification. In terms of runtime-efficiency specification comparison, [68] state that serialization and deserialization speeds vary widely amongst programming languages and data specification implementations and thus is not comparable. For example, a binary specification may outperform a textual specification in one programming language, but the opposite could be true for the same specifications in another programming language. We consider runtime-efficiency to be a consequence of the implementation in the specific programming language rather than a property of the serialization specification. [39] argues that network communication is time-expensive and that the network communication bottleneck *makes computation essentially free in comparison*. It also argues that the computation overhead of making HTTP/1.1 [24] payloads space-efficient using techniques such as data compression can be considered minimal compared to the time-overhead of a low bandwidth network connection, which are still common according to [54]. For these reasons, we focus on studying the space-efficiency characteristics of binary serialization specifications.

JSON [17] documents are typically not sent over the internet uncompressed. [54] found that 92.2% of the most popular websites according to Alexa top 500 websites use HTTP/1.1 [24] compression formats such as GZIP [16]. For this reason, we take into account the impact of data compression formats as we investigate space-efficiency characteristics for the selection of binary serialization specifications.

1.1 Space-efficiency Benchmark Study

Given the JSON-compatible schema-less and schema-driven binary serialization specifications studied in [71], this benchmark aims to answer the following set of research questions:

- **Q1**: How do JSON-compatible schema-less binary serialization specifications compare to JSON in terms of space-efficiency?
- **Q2**: How do JSON-compatible schema-driven binary serialization specifications compare to JSON and JSON-compatible schema-less binary serialization specifications in terms of space-efficiency?
- **Q3**: How do JSON-compatible sequential binary serialization specifications compare to JSON-compatible pointer-based binary serialization specifications in terms of space-efficiency?
- **Q4**: How does compressed JSON compares to uncompressed and compressed JSON-compatible binary serialization specifications?

1.2 Contributions

This paper presents and discusses a space-efficiency benchmark of the 13 JSON-compatible binary serialization specifications studied in [71] using a dataset of 27 methodically JSON documents. We believe this work is the first of its kind to produce a comprehensive, reproducible, extensible and open-source space-efficiency benchmark of JSON-compatible binary serialization specifications that considers a large and representative input dataset of real-world JSON documents across industries and takes data compression into consideration. While producing this benchmark study, we identified a lack of an industry-standard automated software to define JSON-compatible space-efficiency benchmarks. As a solution, we designed and implemented an extensible, automated and deterministic benchmark platform to declare JSON input documents, declare JSON-compatible serialization specifications written in arbitrary programming languages, declare data compression formats, extract raw and aggregate statistical resulting data and generate bar plots and box plots to visualize the results. The benchmark software is publicly-available on GitHub [2] under the Apache-2.0 software license. The benchmark runs on the cloud using GitHub Actions [3] and the results are automatically published to `https://www.jviotti.com/binary-json-size-benchmark/`.

[2] `https://github.com/jviotti/binary-json-size-benchmark`

[3] `https://github.com/features/actions`

Through the process of conducting a literature review of space-efficiency benchmarks involving JSON-compatible serialization specifications, we identified a lack of a methodical approach for selecting representative sets of input JSON documents for benchmarking purposes. We believe this work is the first of its kind to introduce a formal tiered taxonomy for JSON documents (see section 4) consisting of 36 categories as a common basis to class JSON documents based on their size, type of content, characteristics of their structure and redundancy criteria. As part of the taxonomy definition, we developed a publicly-available companion web application called *JSON Stats* to automatically categorize JSON documents according to the taxonomy. This online tool is available at `https://www.jsonbinpack.org/stats/` and its source code is publicly-available on GitHub [4] under the Apache-2.0 software license [5]. Refer to Figure 10 for a screenshot of the tool in action.

1.3 Paper Organization

This paper is organized as follows: in section 1, we motivate the need for space-efficiency and outline the proposed benchmark study. In section 2, we summarize existing literature on space-efficiency benchmark involving the selection of binary serialization specifications: ASN.1 [56], Apache Avro [27], Microsoft Bond [43], Cap'n Proto [69], FlatBuffers [66], Protocol Buffers [31], and Apache Thrift [60], BSON [42], CBOR [5], FlexBuffers [67], MessagePack [29], Smile [55], and UBJSON [7]. We found several aspects of the existing literature lacking merit when it comes to serialization specifications coverage, reproducibility and representativity, the role of data compression and given obsolete versions of certain binary serialization specifications. In section 3, we introduce the SchemaStore dataset, an Apache-2.0 licensed collection of JSON Schema [75] documents and make extensive use of its test suite which consists of over 400 real-world JSON [17] documents matching their respective schemas across various industries. In section 4, we introduce a tiered taxonomy for JSON documents based on the size, dominant content, structure and redundancy characteristics across three main categories: *Tier 1 Minified* < 100 *bytes*, *Tier 2 Minified* $\geq 100 < 1000$ *bytes*, and *Tier 3 Minified* ≥ 1000 *bytes*. We built and published a free-to-use online tool to automatically categorize JSON documents according to the taxonomy that generates related summary statistics. In section 5, we introduce our methodology to extend the body of literature and demonstrate what needs to be done to produce a fair benchmark. In Figure 6, we provide comprehensive results along with their corresponding plots and summary statistics for 27 test cases. In section 7, we demonstrate all the levels we have adhered to in the interest of fairness and reproducibility. In section 8, we engage in a reflective discussion detailing the conclusions of our benchmark study.

2 Related Literature

The existing space-efficiency benchmarks involving the binary serialization specifications discussed in [71] are summarized in Table 1. As demonstrated in Table 1, schema-less binary data serialization specifications such as BSON [42], CBOR [5], MessagePack [29], and Smile [55] tend to produce smaller messages than JSON (up to 63% size reduction compared to JSON). However, there are some exceptions. For instance, [48] and [47] found that BSON [42] and CBOR [5] tend to produce larger bit-strings than JSON [17] for a subset of their input data (up to 32% larger than JSON).

In comparison to schema-less serialization specifications, as demonstrated in Table 1, schema-driven serialization specifications such as Protocol Buffers [31], Apache Thrift [60], Apache Avro [27] tend to produce bit-strings that are up to 95% smaller than JSON. However, in this case, there are also exceptions. For instance, [47] and [68] found the MessagePack [29] schema-less binary serialization specification to be space-efficient in comparison to Protocol Buffers [31] and Apache Avro [27] (up to 23% size reduction). Similarly, [32] and [4] found MessagePack [29] to be space-efficient in comparison to Protocol Buffers [31] and FlatBuffers [66] for certain cases.

Data compression is another approach to achieve space efficiency. [30] conclude that compressed JSON is space-efficient in comparison to both compressed and uncompressed Protocol Buffers. [47] conclude that compressed JSON is space-efficient in comparison to Protocol Buffers, Smile [55], compressed and uncompressed CBOR [5], and compressed and uncompressed BSON [42]. However, Apache Avro [27], MessagePack [29], compressed Protocol Buffers, and compressed Smile are space-efficient in comparison to compressed JSON.

[4] `https://github.com/jviotti/jsonbinpack`

[5] `https://www.apache.org/licenses/LICENSE-2.0.html`

2.1 Shortcomings

We found several aspects of the existing literature to be insufficient, leading to gaps in the JSON-compatible binary serialization space-efficiency benchmark literature for the following reasons:

Coverage of Serialization Specifications. The binary serialization specifications covered by existing benchmarks are Protocol Buffers [31], MessagePack [29], FlatBuffers [66], and to a lesser extent: Apache Avro [27], Apache Thrift [60], BSON [42], CBOR [5] and Smile [55]. Our previous work [71] also discusses ASN.1 [56], Microsoft Bond [43], Cap'n Proto [69], FlexBuffers [67], and UBJSON [7]. To the best of our knowledge, these have not been considered by existing space-efficiency benchmark literature.

Reproducibility and Representativity. Neither [30], [48], [47], [53], [36], nor [50] disclose the JSON [17] documents or schema definitions used to arrive at the result of their space-efficiency benchmarks. Therefore, it is not possible to corroborate their findings or contextualize their results. The publications that disclose the input JSON [17] documents are [41] and [40]. However, they are limited in scope as they consider a single JSON document in each of their papers. Other publications disclose schema definitions of varying formality that describe the JSON documents used as part of their benchmarks. Of those, [72] and [49] are concerned with one type of JSON document, [63] and [22] are concerned with two types of JSON documents, and [32] and [4] are concerned with three types of JSON documents. Therefore, we consider the results from these publications to be either not reproducible or not representative of the variety of JSON documents that are widely used in practice across different industries.

Data Compression. We found two publications that take data compression into account: [30] and [47]. However, they are limited in scope as these papers discuss only the GZIP [16] data compression format and there is no mention of the implementation used and the compression level that GZIP is configured with.

Out-of-date. Some of the existing benchmarks measure obsolete versions of certain binary serialization specifications. For example, the Protocol Buffers [31] version 3 was first released in 2014 [6]. However, there are a number of benchmark publications released before that year that discuss the now-obsolete Protocol Buffers version 2 [30] [72] [41] [63] [22].

[6] `https://github.com/protocolbuffers/protobuf/releases/tag/v3.0.0-alpha-1`

Table 1: A list of space-efficiency benchmark publications that involve JSON [17] and/or a subset of the binary serialization specifications dicussed in [71]. The third column summarises the benchmark conclusions. In this table, a serialization specification is *greater than* another serialization specification if it produced larger bit-strings in the respective publication findings.

Publication	Year	Conclusion
Impacts of data interchange formats on energy consumption and performance in smartphones [30]	2011	JSON > Protocol Buffers > Protocol Buffers with GZIP > JSON with GZIP
Evaluation of Protocol Buffers as Data Serialization Format for Microblogging Communication [72]	2011	JSON > Protocol Buffers
Performance evaluation of object serialization libraries in XML, JSON and binary formats [41]	2012	JSON > Apache Thrift > Protocol Buffers > Apache Avro
A comparison of data serialization formats for optimal efficiency on a mobile platform [63]	2012	JSON > Apache Thrift > Protocol Buffers
Google protocol buffers research and application in online game [22]	2013	JSON > Protocol Buffers
Integrating a system for symbol programming of real processes with a cloud service [40]	2015	JSON > MessagePack
Performance evaluation of using Protocol Buffers in the Internet of Things communication [48]	2016	**In most cases**: JSON > BSON > Protocol Buffers. **However, in some cases**: BSON > JSON > Protocol Buffers
Smart grid serialization comparison: Comparison of serialization for distributed control in the context of the Internet of Things [47]	2017	BSON > CBOR > JSON > BSON with GZIP > Smile > Protocol Buffers > CBOR with GZIP > JSON with GZIP > Apache Avro > Protocol Buffers with GZIP > Smile with GZIP > MessagePack > Apache Avro with GZIP > MessagePack with GZIP
Binary Representation of Device Descriptions: CBOR versus RDF HDT [53]	2018	JSON > CBOR
Evaluating Serialization for a Publish-Subscribe Based Middleware for MPSoCs [32]	2018	FlatBuffers > Protocol Buffers > MessagePack
Performance Evaluation of Java, JavaScript and PHP Serialization Libraries for XML, JSON and Binary Formats [68]	2018	JSON > MessagePack > Protocol Buffers > Apache Avro
Analytical assessment of binary data serialization techniques in IoT context (evaluating protocol buffers, flat buffers, message pack, and BSON for sensor nodes) [4]	2019	**For numeric and mixed data**: BSON > FlatBuffers > MessagePack > Protocol Buffers. **For textual data**: FlatBuffers > BSON > MessagePack > Protocol Buffers
Enabling Model-Driven Software Development Tools for the Internet of Things [36]	2019	JSON > FlatBuffers
Flatbuffers Implementation on MQTT Publish/Subscribe Communication as Data Delivery Format [49]	2019	JSON > FlatBuffers
Performance Comparison of Messaging Protocols and Serialization Formats for Digital Twins in IoV [50]	2020	JSON > FlatBuffers > Protocol Buffers

3 The SchemaStore Dataset

SchemaStore [7] is an Apache-2.0-licensed [8] open-source collection of over 300 JSON Schema [74] documents which describe popular JSON-based [17] formats such as CityJSON [65] and JSON Patch [9]. The SchemaStore API can be integrated with code editors to offer auto-completion and validation when writing JSON documents. The SchemaStore project was started by Mads Kristensen [9] in 2014 while working as a Senior Program Manager focused on the Visual Studio IDE at Microsoft. For the purpose of benchmarking, we will make use of SchemaStore's extensive test suite which consists of over 400 real-world JSON documents matching the respective schemas [10]. This paper refers to commit hash `0b6bd2a08005e6f7a65a68acaf3064d6e2670872` of the SchemaStore repository hosted on GitHub [11]. We believe that the SchemaStore test suite is a good representation of the set of JSON documents used across industries.

```json
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Apple Universal Link, App Site Association",
  "type": "object",
  "properties": {
    "applinks": {
      "type": "object",
      "properties": {
        "apps": {
          "enum": [ [] ],
          "description": "Must be an empty array"
        },
        "details": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "appID": {
                "type": "string",
                "description":
                  "The value of the appID key is the team ID or app ID prefix, followed by the bundle ID"
              },
              "paths": {
                "type": "array",
                "items": {
                  "type": "string",
                  "description":
                    "Ordered list of paths to open in the mobile app. Prefix with 'NOT ' to remove a path"
                }
              }
            }
          }
        }
      },
      "required": [ "apps", "details" ]
    }
  },
  "required": [ "applinks" ]
}
```

Figure 1: An example JSON Schema Draft 4 [20] document from SchemaStore [14] that describes an Apple Associated Domain file [15] to associate an iOS app and a website.

[7] `https://www.schemastore.org`
[8] `http://www.apache.org/licenses/LICENSE-2.0.html`
[9] `https://github.com/madskristensen`
[10] `https://github.com/SchemaStore/schemastore/tree/master/src/test`
[11] `https://github.com/SchemaStore/schemastore`

```json
{
  "applinks": {
    "apps": [],
    "details": [
      {
        "appID": "9JA89QQLNQ.com.apple.wwdc",
        "paths": [ "/wwdc/news/", "/videos/wwdc/2015/*" ]
      },
      {
        "appID": "ABCD1234.com.apple.wwdc",
        "paths": [ "*" ]
      }
    ]
  }
}
```

Figure 2: An example JSON [17] document that matches the schema definition from Figure 1 taken from SchemaStore's test suite [17].

4 A Taxonomy of JSON Documents

Serializing two data structures that match the same schema definition but consist of different values is likely to result in similar byte sizes. However, serializating two data structures with the same values but different structures may produce diverse results, even when utilising the same serialization specification. Therefore, we conclude that the structure and the type of content affects the size of the serialized bit-strings more than the actual values. Under this assumption, to produce a representative size benchmark, it is essential to measure binary serialization specifications using a set of JSON [17] documents that differ in structure, type of content and size.

To solve the input data selection problem, it is required to have a process to categorize JSON [17] documents depending on such characteristics. In this way, we present a taxonomy consisting of 36 categories listed in Table 2. The taxonomy qualifies JSON documents based on their size, type of content, nesting, structural, and redundancy characteristics. While most JSON documents in practice are objects or arrays, this taxonomy is also applicable to JSON documents consisting of single scalar values and strings. We hope that this taxonomy forms a common basis to talk about JSON documents in a high-level manner beyond the benchmarking problem.

4.1 Size

In order to categorize JSON documents in a sensible manner using a small set of size categories, we first calculate the byte-size distribution of the JSON documents in the SchemaStore test suite introduced in section 3. The results are illustrated in Figure 4. Based on these results, we group JSON documents into three categories:

- **Tier 1 Minified $<$ 100 bytes.** A JSON document is in this category if its UTF-8 [14] *minified* form occupies less than 100 bytes [71].
- **Tier 2 Minified $\geq$ 100 $<$ 1000 bytes.** A JSON document is in this category if its UTF-8 [14] *minified* form occupies 100 bytes or more, but less than 1000 bytes [71].
- **Tier 3 Minified $\geq$ 1000 bytes.** A JSON document is in this category if its UTF-8 [14] *minified* form occupies 1000 bytes or more [71].

```
$ node
Welcome to Node.js v12.22.1.
Type ".help" for more information.
> const document = { foo: 'bar' }
undefined
> Buffer.byteLength(JSON.stringify(document), 'utf8')
13
```

Figure 3: The UTF-8 [14] byte-size of a JSON document in *minified* form can be determined using a Node.js [19] interactive REPL session by combining the `JSON.stringify` and the `Buffer.byteLength` functions as demonstrated in this figure. In this example, we determine the size of the JSON document `{ "foo": "bar" }` to be 13 bytes.

4.2 Content Type

The taxonomy categorises a JSON document based on the data types that dominate its content. We calculate this characteristic based on the number of values of a certain data type that a JSON document contains and the byte-size that these data values occupy in the serialized bit-string. We take both of these measures into account as serializing many small instances result in more metadata overhead than serializing a few large instances for a given type.

The JSON [17] serialization specification supports the following data types: *object*, *array*, *boolean*, *string*, *number*, and *null*. We consider objects and arrays to represent *structural* values, strings to represent *textual* values, and numbers to represent *numeric* values. For simplicity, we consider *true*, *false*, and *null* to represent *boolean* values as in three-valued logic [51]. We use this data type categorization to define the *textual weight*, *numeric weight*, and *boolean weight* for a given JSON document.

The weight metrics for a JSON document are based on a common formula where C is the total number of data values in the JSON document and S is the total byte-size of the JSON document in *minified* form [71]:

$$\frac{\frac{K \times 100}{C} \times \frac{B \times 100}{S}}{100} \tag{1}$$

- **Textual Weight.** In this case, K is the number of string values in the JSON document and B is the cummulative byte-size occupied by the string values in the JSON document. The

Table 2: There are 36 categories defined in our JSON documents taxonomy. The second column contains acronyms for each category name. In terms of size, a JSON document can either be *Tier 1 Minified < 100 bytes*, *Tier 2 Minified ≥ 100 < 1000 bytes*, or *Tier 3 Minified ≥ 1000 bytes*. In terms of content, a JSON document can either be *numeric*, *textual*, or *boolean*. Finally, in terms of structure, a JSON document can either be *flat* or *nested*.

Category				**Acronym**
Tier 1 Minified $<$ 100 bytes	Numeric	Redundant	Flat	`Tier 1 NRF`
Tier 1 Minified $<$ 100 bytes	Numeric	Redundant	Nested	`Tier 1 NRN`
Tier 1 Minified $<$ 100 bytes	Numeric	Non-Redundant	Flat	`Tier 1 NNF`
Tier 1 Minified $<$ 100 bytes	Numeric	Non-Redundant	Nested	`Tier 1 NNN`
Tier 1 Minified $<$ 100 bytes	Textual	Redundant	Flat	`Tier 1 TRF`
Tier 1 Minified $<$ 100 bytes	Textual	Redundant	Nested	`Tier 1 TRN`
Tier 1 Minified $<$ 100 bytes	Textual	Non-Redundant	Flat	`Tier 1 TNF`
Tier 1 Minified $<$ 100 bytes	Textual	Non-Redundant	Nested	`Tier 1 TNN`
Tier 1 Minified $<$ 100 bytes	Boolean	Redundant	Flat	`Tier 1 BRF`
Tier 1 Minified $<$ 100 bytes	Boolean	Redundant	Nested	`Tier 1 BRN`
Tier 1 Minified $<$ 100 bytes	Boolean	Non-Redundant	Flat	`Tier 1 BNF`
Tier 1 Minified $<$ 100 bytes	Boolean	Non-Redundant	Nested	`Tier 1 BNN`
Tier 2 Minified $\geq$ 100 $<$ 1000 bytes	Numeric	Redundant	Flat	`Tier 2 NRF`
Tier 2 Minified $\geq$ 100 $<$ 1000 bytes	Numeric	Redundant	Nested	`Tier 2 NRN`
Tier 2 Minified $\geq$ 100 $<$ 1000 bytes	Numeric	Non-Redundant	Flat	`Tier 2 NNF`
Tier 2 Minified $\geq$ 100 $<$ 1000 bytes	Numeric	Non-Redundant	Nested	`Tier 2 NNN`
Tier 2 Minified $\geq$ 100 $<$ 1000 bytes	Textual	Redundant	Flat	`Tier 2 TRF`
Tier 2 Minified $\geq$ 100 $<$ 1000 bytes	Textual	Redundant	Nested	`Tier 2 TRN`
Tier 2 Minified $\geq$ 100 $<$ 1000 bytes	Textual	Non-Redundant	Flat	`Tier 2 TNF`
Tier 2 Minified $\geq$ 100 $<$ 1000 bytes	Textual	Non-Redundant	Nested	`Tier 2 TNN`
Tier 2 Minified $\geq$ 100 $<$ 1000 bytes	Boolean	Redundant	Flat	`Tier 2 BRF`
Tier 2 Minified $\geq$ 100 $<$ 1000 bytes	Boolean	Redundant	Nested	`Tier 2 BRN`
Tier 2 Minified $\geq$ 100 $<$ 1000 bytes	Boolean	Non-Redundant	Flat	`Tier 2 BNF`
Tier 2 Minified $\geq$ 100 $<$ 1000 bytes	Boolean	Non-Redundant	Nested	`Tier 2 BNN`
Tier 3 Minified $\geq$ 1000 bytes	Numeric	Redundant	Flat	`Tier 3 NRF`
Tier 3 Minified $\geq$ 1000 bytes	Numeric	Redundant	Nested	`Tier 3 NRN`
Tier 3 Minified $\geq$ 1000 bytes	Numeric	Non-Redundant	Flat	`Tier 3 NNF`
Tier 3 Minified $\geq$ 1000 bytes	Numeric	Non-Redundant	Nested	`Tier 3 NNN`
Tier 3 Minified $\geq$ 1000 bytes	Textual	Redundant	Flat	`Tier 3 TRF`
Tier 3 Minified $\geq$ 1000 bytes	Textual	Redundant	Nested	`Tier 3 TRN`
Tier 3 Minified $\geq$ 1000 bytes	Textual	Non-Redundant	Flat	`Tier 3 TNF`
Tier 3 Minified $\geq$ 1000 bytes	Textual	Non-Redundant	Nested	`Tier 3 TNN`
Tier 3 Minified $\geq$ 1000 bytes	Boolean	Redundant	Flat	`Tier 3 BRF`
Tier 3 Minified $\geq$ 1000 bytes	Boolean	Redundant	Nested	`Tier 3 BRN`
Tier 3 Minified $\geq$ 1000 bytes	Boolean	Non-Redundant	Flat	`Tier 3 BNF`
Tier 3 Minified $\geq$ 1000 bytes	Boolean	Non-Redundant	Nested	`Tier 3 BNN`

double quotes surrouding string values are considered part of the byte-size occupied by a string. Therefore, a string value encoded in a UTF-8 [14] JSON document occupies at least $2 + N$ bytes where N corresponds to number of code-points in the string.

- **Numeric Weight.** In this case, K is the number of numeric values in the JSON document and B is the cummulative byte-size occupied by the numeric values in the JSON document. Each numeric digit and auxiliary characters such as the minus sign ($-$) and the period (.) for representing real numbers count towards the byte-size of the numeric value.
- **Boolean Weight.** In this case, K is the number of boolean values in the JSON document and B is the cummulative byte-size occupied by the boolean values in the JSON document. The UTF-8 [14] JSON encoding represents *true* using 4 bytes, *false* using 5 bytes, and *null* using 4 bytes.

We rely on the previous weight definitions to provide the content type taxonomy for JSON documents based on whether they are *textual*, *numeric*, or *boolean*. Given an input JSON document, consider W_t, W_n, and W_b to represent its textual, numeric and boolean weights, respectively:

- **Textual.** A JSON document is textual if $W_t \geq W_n \geq W_b$.
- **Numeric.** A JSON document is numeric if $W_n \geq W_t \geq W_b$.
- **Boolean.** A JSON document is numeric if $W_b \geq W_t \geq W_n$.

If two or more of the content type weight values are equal and greater than the rest, such JSON document is considered to hold more than one type of content qualifier. For example, if $W_t = W_n$ and $W_t > W_b$, then the JSON document is equally considered *textual* and *numeric*.

The results of executing this aspect of the taxonomy on the SchemaStore test suite introduced in section 3 are shown in Figure 5.

4.3 Redundancy

The taxonomy measures redundancy as the percentage of values in a given JSON document that are duplicated taking scalar and composite data types into account.

In comparison to schema-less serialization formats, schema-driven serialization formats make use of schema definitions to avoid encoding object keys. This taxonomy is designed to aid in categorizing JSON documents based on characteristics that impact data serialization. For these reasons, the

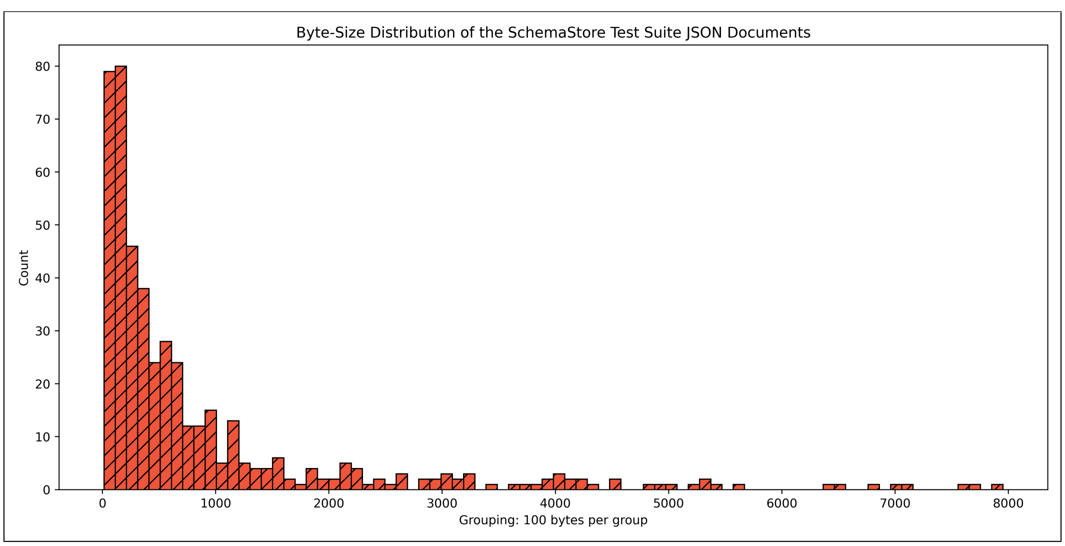

Figure 4: The byte-size distribution (in 100 byte groups up to 8000 bytes for illustration purposes) of the 480 JSON documents present in the SchemaStore test suite introduced in section 3. Most JSON documents weigh less than 1000 bytes. The largest JSON document weighs 545392 bytes [21].

number of duplicated object keys in the redundancy metric is irrelevant for the schema-driven subset of the selection of binary serializion formats and is not taken into account.

Let *JSON* be the set of JSON documents as defined in the data model introduced by [6] with the exception that the $[*]$ object operator results in a *sequence*, instead of a set, of values in the given JSON object. Consider a new $[\&]$ operator defined using the Z formal specification notation [35] that results in the flattened sequence of atomic and compositional structure values of the given JSON document:

$$
\begin{array}{|lr}
\_[\&] : \mathit{JSON} \rightarrow \operatorname{seq} \mathit{JSON} & \\
\hline
\forall J : \mathit{JSON} \bullet & \\
\quad J[\&] = \langle J \rangle \frown J[*][0][\&] \frown \ldots \frown J[*][\#J][\&] & \text{if } J \text{ is an object} \\
\quad J[\&] = \langle J \rangle \frown J_0[\&] \frown J'[\&] & \text{if } J \text{ is an array} \\
\quad J[\&] = \langle J \rangle & \text{otherwise}
\end{array}
$$

Using this operator, R_J is defined as the percentage of duplicate values in the JSON document J:

$$
R_J = \frac{(\#J[\&] - \#\{v \mid v \; in \; J[\&]\}) \times 100}{\#J[\&]} \tag{2}
$$

In order to categorize JSON documents in a sensible manner, the taxonomy distinguishes between *redundant* JSON documents and *non-redundant* JSON documents. The redundancy distribution of the JSON documents in the SchemaStore test suite introduced in section 3 is computed in Figure 6. Using these results, this taxonomy aspect is defined as follows:

- **Redundant.** A JSON document J is redundant if $R_J \geq 25\%$
- **Non-Redundant.** A JSON document J is redundant if $R_J < 25\%$

4.4 Structure

[6] propose that connected acyclic undirected graphs which resembles a tree structure are a natural representation for JSON documents as exemplified in Figure 7. We use the following definitions that define two features associated with the tree: *height* and *level*.

Definition 1. *The height of a node is the number of edges on the longest downward path between that node and a leaf. The height of a tree is the height of its root.*

Definition 2. *The level of a node is defined by 1 + the number of connections between the node and the root. The level is depth + 1.*

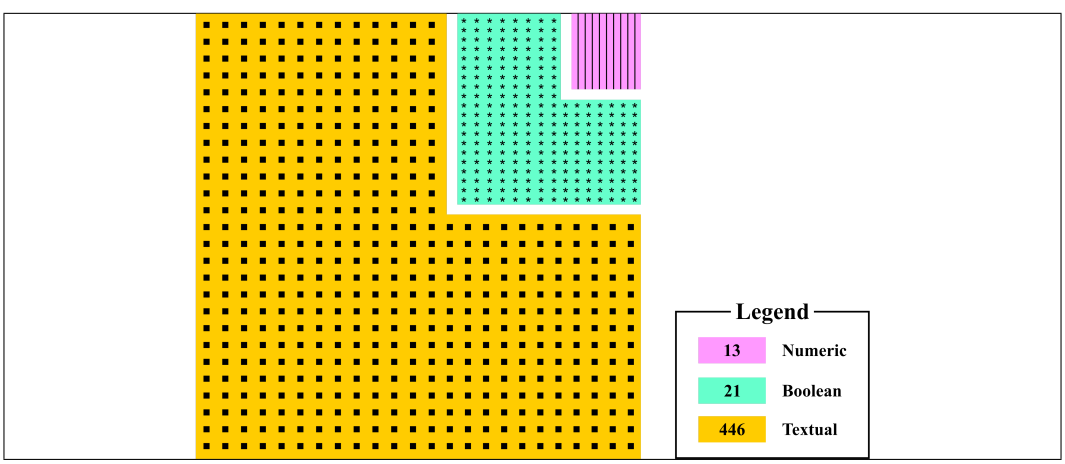

Figure 5: Out of the 480 JSON documents in the SchemaStore test suite introduced in section 3, 446 are textual, 21 are boolean, and 13 are numeric.

Using the definition of height, we extrapolate that the height of the tree determines the height of a given JSON document. Using the definition of level, we extrapolate that the *size* of a level in the tree equals the sum of the byte-size of every textual, numeric, and boolean values whose nodes have the corresponding level. Therefore, the *largest level* is the level with the highest size without taking into account the subtree at depth 0.

The *nesting weight* of a JSON document J, referred to as N_J, is defined as the product of its height and largest level minus 1. We do not consider the byte-size overhead introduced by compositional structures (object and array) in the JSON document as we found that it is highly correlated to its nesting characteristics.

$$N_J = \text{height} \times \text{largest level} - 1 \tag{3}$$

In order to categorize JSON documents in a sensible manner, the taxonomy distinguishes between *flat* JSON documents and *nested* JSON documents. The nesting weight distribution of the JSON

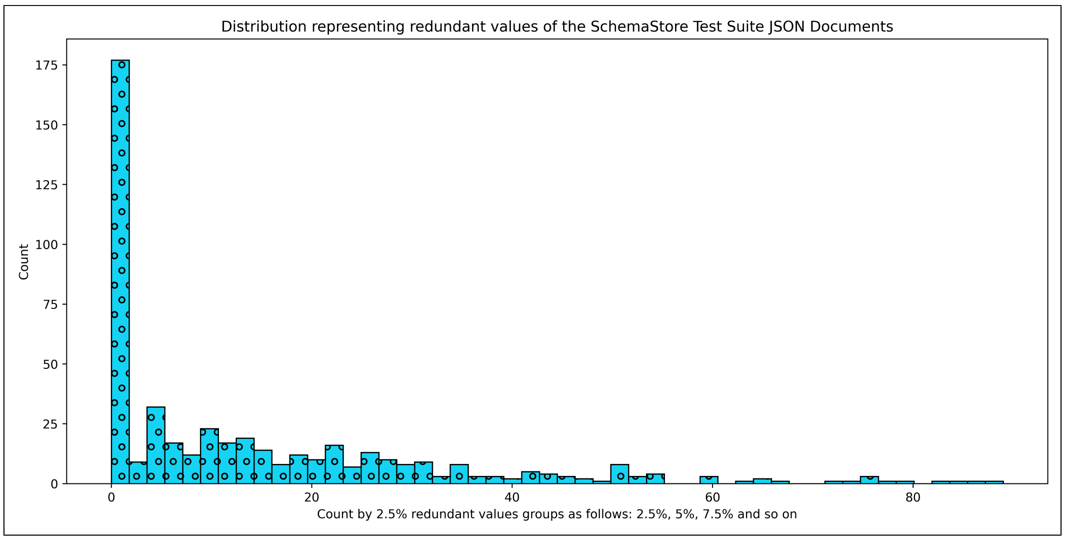

Figure 6: Distribution representing redundant values. First, we calculate the percentage of redundant values of the 480 JSON documents present in the SchemaStore test suite introduced in section 3 and second, we count them by 2.5% redundant values groups as follows: 2.5%, 5%, 7.5% and so on. Most JSON documents are strictly non-redundant. However there are instances of almost every 2.5% redundancy groups in the plot. The most redundant JSON document has a value redundancy of 88.8% [23].

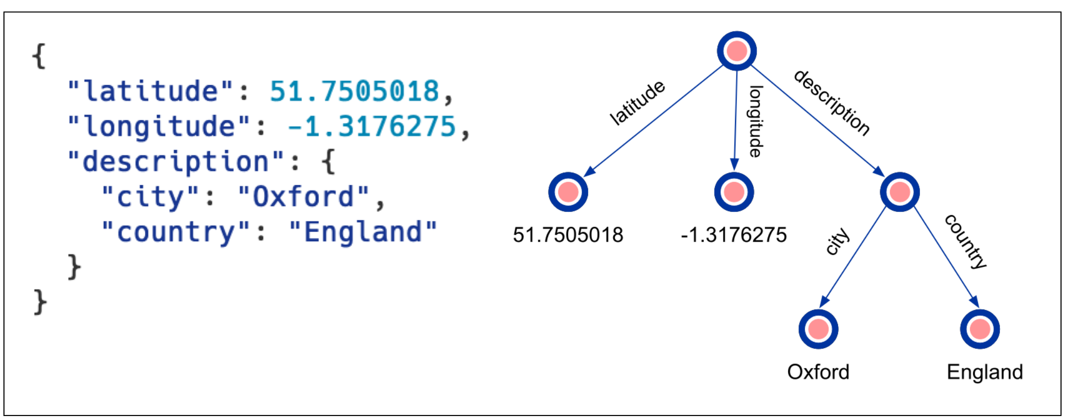

Figure 7: An example JSON document and its corresponding connected acyclic undirected graph representation.

documents in the SchemaStore test suite introduced in section 3 is computed in Figure 8. Using these results, this taxonomy aspect is defined as follows:

- **Flat.** A JSON document J is *flat* if N_J less than the empirically-derived threshold integer value 10.
- **Nested.** A JSON document J is *nested* if N_J is greater than or equal to the empirically-derived threshold integer value 10.

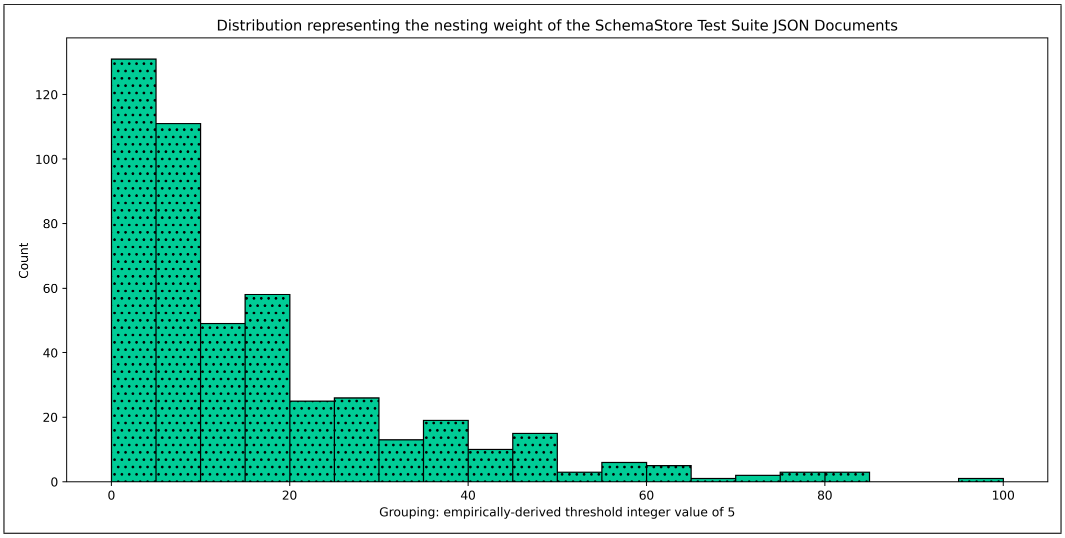

Figure 8: The nesting weight distribution of the 480 JSON documents present in the SchemaStore test suite introduced in section 3 grouped by the empirically-derived threshold 5. Most JSON documents have a nesting weight of under 20. However, there are JSON documents with a nesting weight of up to 100 [25].

4.5 Demonstration

To demonstrate our conceptual taxonomy, we apply it to the JSON document listed in Figure 9. Table 3 provides a breakdown of this JSON document including information such as the number of edges and byte size of every valid JSON Pointer path [10]. This document is a *Tier 2 Minified* $\geq$ *100* $<$ *1000 bytes*, *numeric*, *non-redundant*, and *nested* (SNNN according to Table 2) JSON document:

- **Tier 2 Minified $\geq$ 100 $<$ 1000 bytes.** The size of the JSON document is 184 bytes. 184 is greater than 100 but less than 1000, therefore the JSON document from Figure 9 is *Tier 2 Minified* $\geq$ *100* $<$ *1000 bytes* according to the taxonomy.
- **Numeric.** Table 3 shows that the JSON document has 24 values corresponding to its set of valid JSON Pointer [10] paths. Of those, 7 (29.16%) are numeric, 3 (12.5%) are textual, and 4 (16.66%) are boolean. Out of the 184 total bytes from the JSON document, 24 bytes (13.04%) correspond to numeric values, 14 bytes (7.60%) correspond to textual values, and 17 bytes (9.23%) correspond to boolean values. The numeric weight is $29.16 \times 13.04/100 = 3.80$, the textual weight is $12.5 \times 7.60/100 = 0.95$, and the boolean weight is $16.66 \times 9.23/100 = 1.53$. 3.80 is greater than 0.95 and 1.53, therefore the JSON document from Figure 9 is *numeric* according to the taxonomy.
- **Non-Redundant.** The JSON document consists of 24 values. Out of those, the numeric value 1 appears in the JSON Pointer [10] paths `/days/0`, `/days/1`, and `/days/2`. The textual value *ox03* appears at `/data/0/name` and `/data/2/name`. Similarly, the boolean value *true* appears at `/data/0/staff` and `/data/2/staff`. Furthermore, the objects `/data/0` and `/data/2` are equal. Therefore, only 19 out of the 24 values in the JSON document are unique. We conclude that only 5 (20.83%) of its values are redundant, so the JSON document from Figure 9 is *non-redundant* according to the taxonomy.

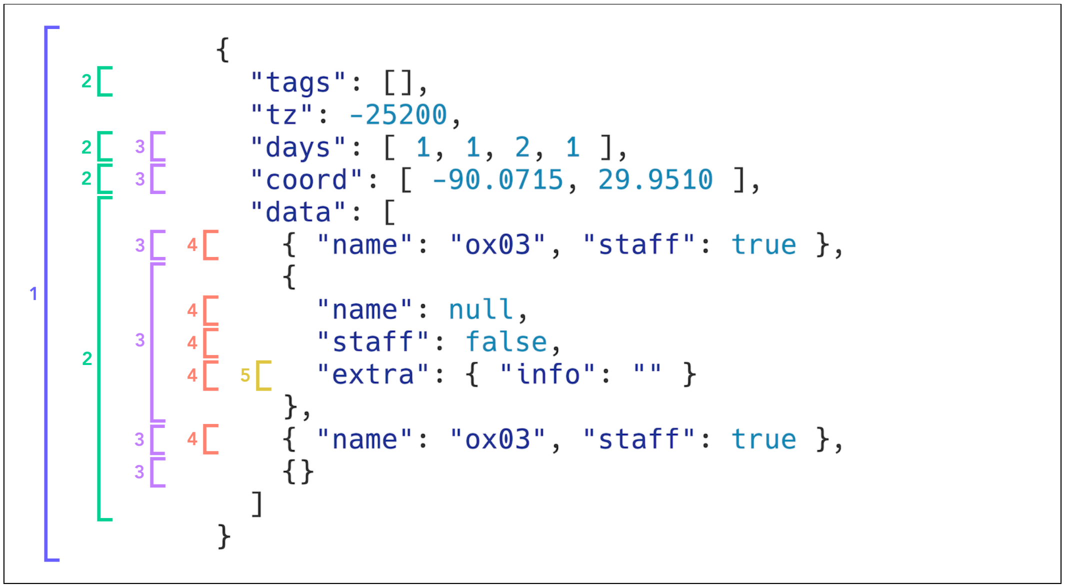

Figure 9: An example *Tier 2 Minified ≥ 100 < 1000 bytes, numeric, non-redundant, and nested* JSON document taken from our previous work [71]. The annotations at the left highlight each level in the JSON document. The height of this document is $(5 - 1 = 4)$ (the highest level minus 1).

- **Nested.** The height is 4, awarded to the pointer `/data/1/extra/info`. We calculate the byte-size of each level by adding the byte-size of each non-structural value in such level. Level 2 occupies 6 bytes, level 3 occupies 18 bytes, level 4 occupies 29 bytes, and level 5 occupies 2 bytes, so level 4 is the largest level. The nesting weight of the JSON document is $4 \times (4 - 1) = 12$ (the height multiplied by the largest level minus 1). 12 is greater than 10, therefore the JSON document from Figure 9 is *nested* according to the taxonomy.

4.6 JSON Stats Analyzer

We built and published a free-to-use online tool at `https://www.jsonbinpack.org/stats/` to automatically categorize JSON documents according to the taxonomy defined in this section and provide summary statistics. Figure 10 demonstrates the summary statistics analyzed for the *Tier 2 Minified ≥ 100 < 1000 bytes, numeric, non-redundant, and nested* JSON document from Figure 9.

The tool is developed using the TypeScript [26] programming language, the CodeMirror [27] open-source embeddable web editor, and the Tailwind CSS [28] open-source web component framework. The web application is deployed to the GitHub Pages [29] free static-hosting service.

[26] `https://www.typescriptlang.org`
[27] `https://codemirror.net`
[28] `https://tailwindcss.com`
[29] `https://pages.github.com`

Table 3: A breakdown of the JSON document from Figure 9 in terms of its valid JSON Pointer [10] paths, value type, level , byte-size, and redundancy.

JSON Pointer	Type	Level	Byte-size	Same As
`/`	Structural	1	184	
`/tags`	Structural	2	2	
`/tz`	Numeric	2	6	
`/days`	Structural	2	9	
`/days/0`	Numeric	3	1	`/days/1`, `/days/3`
`/days/1`	Numeric	3	1	`/days/0`, `/days/3`
`/days/2`	Numeric	3	1	
`/days/3`	Numeric	3	1	`/days/0`, `/days/1`
`/coord`	Structural	2	17	
`/coord/0`	Numeric	3	8	
`/coord/1`	Numeric	3	6	
`/data`	Structural	2	110	
`/data/0`	Structural	3	28	`/data/2`
`/data/0/name`	Textual	4	6	`/data/2/name`
`/data/0/staff`	Boolean	4	4	`/data/2/staff`
`/data/1`	Structural	3	47	
`/data/1/name`	Boolean	4	4	
`/data/1/staff`	Boolean	4	5	
`/data/1/extra`	Structural	4	11	
`/data/1/extra/info`	Textual	5	2	
`/data/2`	Structural	3	28	`/data/0`
`/data/2/name`	Textual	4	6	`/data/0/name`
`/data/2/staff`	Boolean	4	4	`/data/0/staff`
`/data/3`	Structural	3	2	

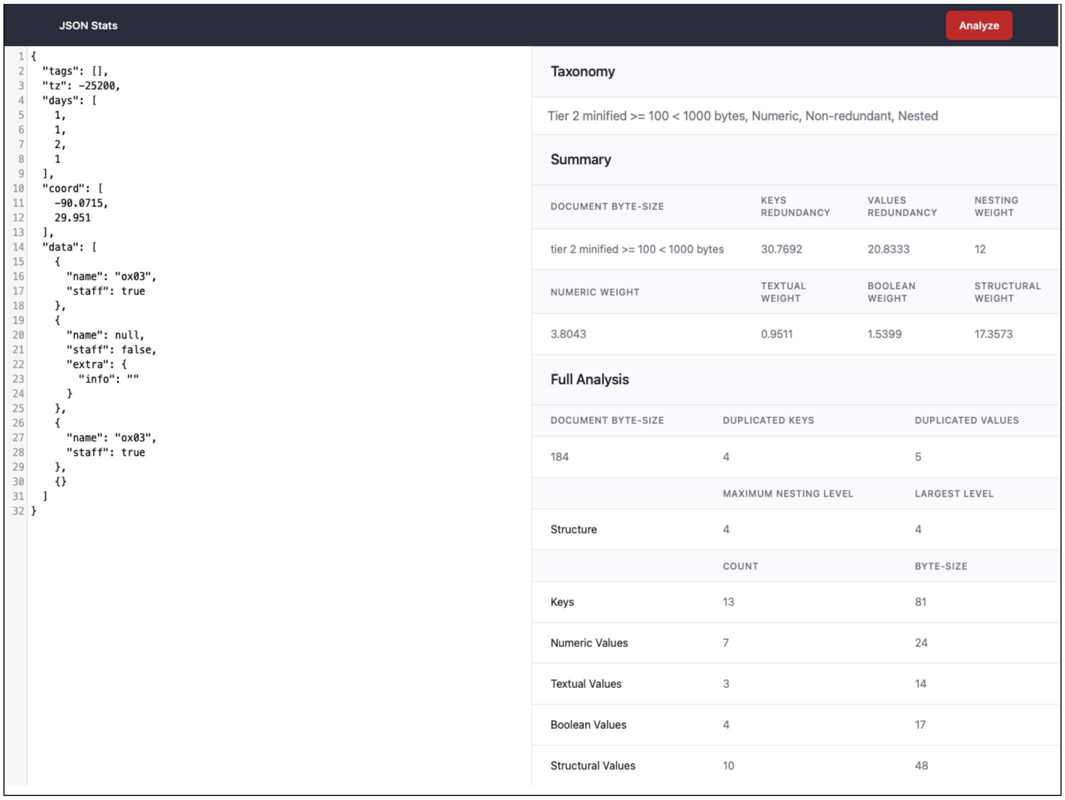

Figure 10: A screenshot of the online tool published at `https://www.jsonbinpack.org/stats/` analyzing the JSON document listed in Figure 9.

Discussion. The document under analysis in Figure 10 is displayed in the embedded text editor on the left side of the screen. The analysis results are present on the right side of the screen and are generated after pressing the red *Analyze* button on the top right corner. The *Taxonomy* section of the analysis table shows that the document is a *Tier 2 Minified* $\geq$ *100* $<$ *1000 bytes, numeric, non-redundant, and nested* document according to the taxonomy. The *Summary* section shows the intermediary results of the size, content type, redundancy and structural statistics introduced in subsection 4.1, subsection 4.2, subsection 4.3 and subsection 4.4, respectively. The *Full Analysis* section shows all the intermediary values used throughout every calculation.

5 Methodology

Our approach to extend the body of literature through a space-efficiency benchmark of JSON-compatible binary serialization specifications is based on the following methodology:

1. **Input Data.** Select a representative set of real-world JSON [17] documents across industries according to the taxonomy defined in section 4.
2. **Serialization Specifications.** Drawing on research from [71], list the set of JSON-compatible schema-less and schema-driven binary serialization specifications to be benchmarked along with their respective encodings and implementations.
3. **Compression Formats.** Select a set of popular lossless data compression formats along with their respective implementations. These compression formats will be used to compress the input JSON [17] documents and bit-strings generated by the selection of binary serialization formats.
4. **Schema Definitions.** Write schema definitions for each combination of input JSON [17] document and selected schema-driven binary serialization specification.
5. **Benchmark.** Serialize each JSON [17] document using the selection of binary serialization specifications. Then, deserialize the bit-strings and compare them to the original JSON [17] documents to test that there is no accidental loss of information.
6. **Results.** Measure the byte-size of the JSON [17] documents and bit-strings generated by each binary serialization specification in uncompressed and compressed form using the selection of data compression formats.
7. **Conclusions.** Discuss the results to identify space-efficient JSON-compatible binary serialization specifications and the role of data compression in increasing space-efficiency of JSON [17] documents.

5.1 Input Data

Figure 11 categorizes the JSON [17] documents from the SchemaStore test suite introduced in section 3 according to the taxonomy defined in section 4. The SchemaStore test suite does not contain JSON [17] documents that match 9 out of the 36 categories defined in the taxonomy, particularly in the *Tier 3 Minified $\geq$ 1000 bytes* size category which is dominated by *textual* JSON documents. We embrace these results to conclude that the missing categories do not represent instances of JSON [17] documents that are commonly encountered in practice. The missing categories are the following:

- Tier 2 Minified $\geq 100 < 1000$ bytes Numeric Redundant Flat (SNRF)
- Tier 2 Minified $\geq 100 < 1000$ bytes Boolean Redundant Nested (SBRN)
- Tier 2 Minified $\geq 100 < 1000$ bytes Boolean Non-Redundant Nested (SBNN)
- Tier 3 Minified ≥ 1000 bytes Numeric Redundant Nested (LNRN)
- Tier 3 Minified ≥ 1000 bytes Numeric Non-Redundant Flat (LNNF)
- Tier 3 Minified ≥ 1000 bytes Numeric Non-Redundant Nested (LNNN)
- Tier 3 Minified ≥ 1000 bytes Boolean Redundant Nested (LBRN)
- Tier 3 Minified ≥ 1000 bytes Boolean Non-Redundant Flat (LBNF)
- Tier 3 Minified ≥ 1000 bytes Boolean Non-Redundant Nested (LBNN)

We selected a single JSON [17] document from each matching category. The selection of JSON documents is listed in Table 4 and Table 5. Some JSON [17] documents we selected from the SchemaStore test suite, namely *Entry Point Regulation manifest* and *.NET Core project.json*, include a top level *$schema* string property that is not consider in the benchmark. The use of this keyword is a non-standard approach to make JSON [17] documents reference their own JSON Schema [75] definitions. This keyword is not defined as part of the formats that these JSON documents represent in the SchemaStore dataset.

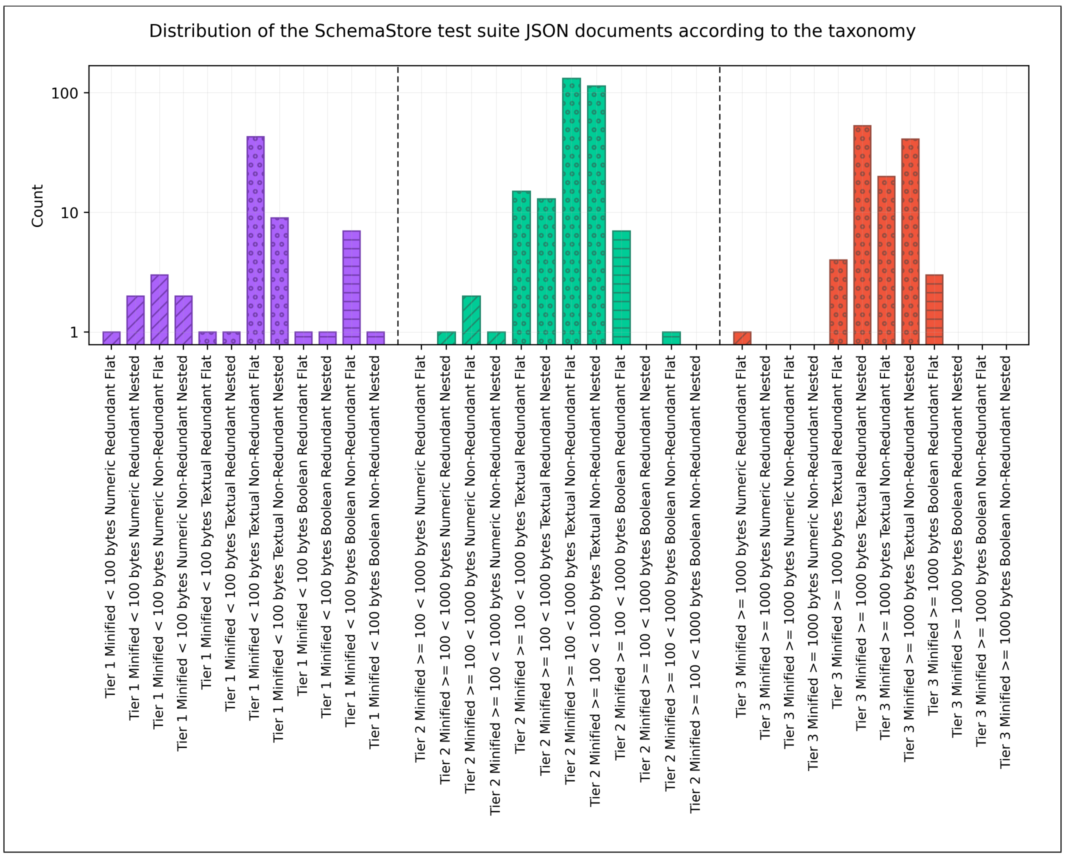

Figure 11: There are 480 JSON [17] documents present in the SchemaStore test suite. These are grouped according to the taxonomy defined in section 4 using a logarithmic y-scale.

Table 4: The JSON [17] documents selected from the SchemaStore test suite introduced in section 3 divided by industry. Each JSON document matches a different taxonomy category defined in section 4. The first column consists of a brief description of the JSON document. The second column contains the path and link to the test case file within the SchemaStore repository. The third column contains the taxonomy categories using the acronyms defined in Table 2. This table is continued in Table 5.

Description	**Test Case Name**	**Category**
	Continuous Integration / Continuous Deliver (CI/CD)	
JSON-e templating engine sort example	`jsone/sort.json`	TNRF
JSON-e templating engine reverse sort example	`jsone/reverse-sort.json`	TNRN
CircleCI definition (blank)	`circleciconfig/version-2.0.json`	TNNF
CircleCI matrix definition	`circleciconfig/matrix-simple.json`	TNNN
SAP Cloud SDK Continuous Delivery Toolkit configuration	`cloud-sdk-pipeline-config-schema/empty.json`	TBRF
TravisCI notifications configuration	`travis/notification-secure.json`	STRF
GitHub Workflow Definition	`github-workflow/919.json`	STNN
	Software Engineering	
Grunt.js "clean" task definition	`grunt-clean-task/with-options.json`	TTRF
CommitLint configuration	`commitlintrc/commitlintrc-test5.json`	TTRN
TSLint linter definition (extends only)	`tslint/tslint-test5.json`	TTNF
TSLint linter definition (multi-rule)	`tslint/tslint-test25.json`	TBRN
CommitLint configuration (basic)	`commitlintrc/commitlintrc-test3.json`	TBNF
TSLint linter definition (basic)	`tslint/tslint-test19.json`	TBNN
ESLint configuration document	`eslintrc/WebAnalyzer.json`	LNRF
NPM Package.json Linter configuration manifest	`npmpackagejsonlintrc/npmpackagejsonlintrc-test.json`	LTRF
.NET Core project.json	`project/EF-project.json`	LTRN
NPM Package.json example manifest	`package/package-test.json`	LTNF

Table 5: Continuation of Table 4.

Description	Test Case Name	Category
Web		
ImageOptimizer Azure Webjob configuration	`imageoptimizer/default.json`	TTNN
Entry Point Regulation manifest	`epr-manifest/official-example.json`	STRN
ECMAScript module loader definition	`esmrc/.esmrc_.json`	SBNF
Nightwatch.js Test Framework Configuration	`nightwatch/default.json`	LBRF
Geospatial		
GeoJSON example JSON document	`geojson/multi-polygon.json`	SNRN
Weather		
OpenWeatherMap API example JSON document	`openweather.current/example.json`	SNNF
OpenWeather Road Risk API example	`openweather.roadrisk/example.json`	SNNN
Publishing		
JSON Feed example document	`feed/microblog.json`	STNF
Open-Source		
GitHub FUNDING sponsorship definition (empty)	`github-funding/ebookfoundation.json`	SBRF
Recruitment		
JSON Resume	`resume/richardhendriks.json`	LTNN

5.2 Serialization Specifications

The selection of schema-driven and schema-less JSON-compatible binary serialization specifications is listed in Table 6 and Table 7. In comparison to our previous work [71], we use ASN-1Step 10.0.2 instead of 10.0.1, Microsoft Bond [43] 9.0.4 instead of 9.0.3, and Protocol Buffers [31] 3.15.3 instead of 3.13.0. None of these version upgrades involve changes to the encodings. Furthermore, we replaced the third-party BSON [42] Python implementation used in [71] with the Node.js official MongoDB implementation. We also replaced the Smile [55] Python implementation used in [71] with a Clojure implementation as we identified issues in the former implementation with respects to floating-point numbers. For example, encoding the floating-point number 282.55 results in 282.549988 when using `pysmile` v0.2.

Finally, both the binary and the packed encoding provided by Cap'n Proto [69] are considered. As described in [71], the packed encoding consists of a basic data compression format officially supported as a separate encoding. These encodings are separately considered to understand the impact of general-purpose data compression on the uncompressed Cap'n Proto [69] variant.

Table 6: The selection of schema-driven JSON-compatible binary serialization specifications based on our previous work [71].

Specification	Implementation	Encoding	License
ASN.1	OSS ASN-1Step Version 10.0.2	PER Unaligned [57]	Proprietary
Apache Avro	Python `avro` (pip) 1.10.0	Binary Encoding [30]with no framing	Apache-2.0
Microsoft Bond	C++ library 9.0.4	Compact Binary v1 [31]	MIT
Cap'n Proto	`capnp` command-line tool 0.8.0	Binary Encoding [32]	MIT
Cap'n Proto	`capnp` command-line tool 0.8.0	Packed Encoding [33]	MIT
FlatBuffers	`flatc` command-line tool 1.12.0	Binary Wire Format [34]	Apache-2.0
Protocol Buffers	Python `protobuf` (pip) 3.15.3	Binary Wire Format [35]	3-Clause BSD
Apache Thrift	Python `thrift` (pip) 0.13.0	Compact Protocol [36]	Apache-2.0

Table 7: The selection of schema-less JSON-compatible binary serialization specifications based on our previous work [71].

Specification	Implementation	License
BSON	Node.js `bson` (npm) 4.2.2	Apache-2.0
CBOR	Python `cbor2` (pip) 5.1.2	MIT
FlexBuffers	`flatc` command-line tool 1.12.0	Apache-2.0
MessagePack	`json2msgpack` command-line tool 0.6 with `MPack` 0.9dev	MIT
Smile	Clojure `cheshire` 5.10.0	MIT
UBJSON	Python `py-ubjson` (pip) 0.16.1	Apache-2.0

[30] `https://avro.apache.org/docs/current/spec.html#binary_encoding`
[31] `https://microsoft.github.io/bond/reference/cpp/compact__binary_8h_source.html`
[32] `https://capnproto.org/encoding.html#packing`
[33] `https://capnproto.org/encoding.html`
[34] `https://google.github.io/flatbuffers/flatbuffers_internals.html`
[35] `https://developers.google.com/protocol-buffers/docs/encoding`
[36] `https://github.com/apache/thrift/blob/master/doc/specs/thrift-compact-protocol.md`

5.3 Schema Definitions

For brevity, this paper does not include the schema definitions for each input JSON document listed in Table 4 and Table 5 for every selected serialization specifications listed in Table 6. The schema definitions can be found on the GitHub repository implemented as part of the benchmark study [37]. Direct links to the corresponding schema definitions are provided along with the benchmark results for each JSON input document.

5.4 Fair Benchmarking

In order to produce a fair benchmark, the resulting bit-strings are ensured to be lossless encodings of the respective input JSON [17] documents. For some binary serialization specifications such as Cap'n Proto [69], providing a schema that only describes a subset of the input data will result in only such subset being serialized and the remaining of the input data being silently discarded. In other cases, a serialization specification may silently coerce an input data type to match the schema definition even at the expense of loss of information. For example, Protocol Buffers [31] may forcefully cast to IEEE 764 32-bit floating-point encoding [26] if requested by the schema even if the input real number can only be represented without loss of precision by using the IEEE 764 *64-bit* floating-point encoding [26].

The implemented benchmark program prevents such accidental mistakes by automatically ensuring that for each combination of serialization specification listed in subsection 5.2 and input JSON document listed in Table 4 and Table 5, the produced bit-strings encode the same information as the respective input JSON document. The automated test consists in serializing the input JSON document using a given binary serialization specification, deserializing the resulting bit-string and asserting that the original JSON document is strictly equal to the deserialized JSON document.

5.5 Compression Formats

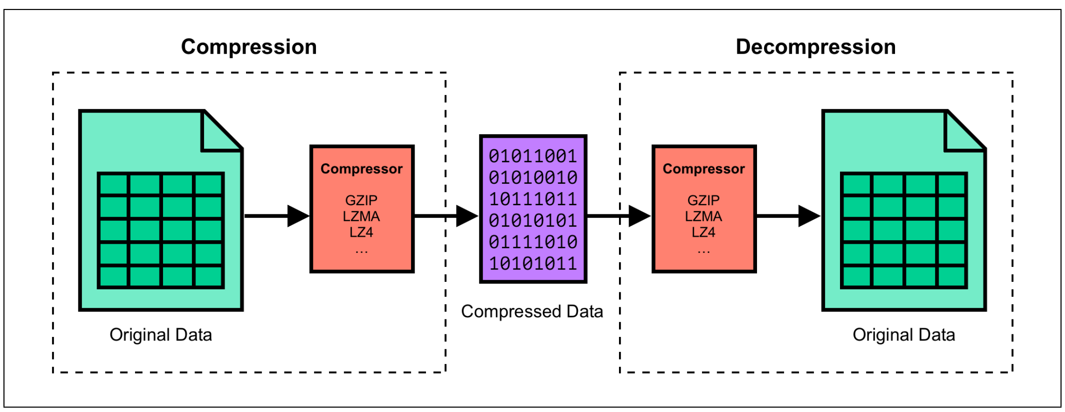

Figure 12: A general-purpose lossless compressor provides a *compression* and a *decompression* process. Compression consists in transforming the original data into a compressed representation of the same information. Decompression reverses the compression process to obtain the unmodified original data from its compressed representation.

We selected the following data compression formats: GZIP (GNU ZIP) [16], LZ4 [38], and Lempel-Ziv-Markov Chain Algorithm (LZMA), a set of compressors commonly used in the context of web services. These three formats are derived from the LZ77 (Lempel-Ziv) [76] dictionary-based coding scheme and are considered general-purpose lossless compressors [37]. The LZ77 [76] algorithm operates by deduplicating multiple occurrences of the same data pattern within certain distance [37] [58], a technique also discussed in more detail in [62]. According to [58], the compression ratios on textual data when using Lempel-Ziv-derived algorithm ranges between 30% and 40% in practice.

[37] `https://github.com/jviotti/binary-json-size-benchmark`

[38] `https://lz4.github.io/lz4/`

Table 8: A high-level view of the differences and similarities between the GZIP, LZ4 and LZMA lossless compression formats.

Compression Format	**GZIP**	**LZ4**	**LZMA**
Differences	Efficient and constant memory usage	High compression and decompression speed	Better compression on large files
Similarities	Based on LZ77 (Lempel-Ziv) [76]		

GZIP (GNU ZIP) [16] is an open-source compressor based on a mixture of the LZ77 [76] and the Huffman [34] coding schemes. GZIP was developed as part of the GNU Project [39] and released in 1992 as a replacement for the UNIX `compress` [40] program. GZIP it is the most widely-used compression format for HTTP/1.1 [25]. [23] study the problem of compressing large amounts of textual and highly-redundant data such as HTML [21] documents and use GZIP as the reference compressor due to its popularity. Their findings show that GZIP has been designed to have a small memory footprint and operate in constant space complexity [58]. These characteristics are possible given that GZIP splits the input data into small blocks of less than 1 MB and compresses each block separately. As a drawback, this approach limits GZIP ability to detect redundancy across blocks and reduces its space-efficiency on larger input data. We consider this drawback to be irrelevant for this benchmark as the largest JSON [8] document present in the SchemaStore test suite introduced in section 3 weights $\tilde{0}.5$ MB as discussed in Figure 4.

LZ4 is an open-source compressor developed by Yann Collet [41] while working at Facebook. LZ4 is a derivative of LZ77 [76]. LZ4 focuses on improving compression and decompression speed by using a hash table data structure for storing reference addresses. The hash table provides constant $\mathcal{O}(1)$ instead of linear $\mathcal{O}(n)$ complexity for match detection [3]. LZ4 is a core part of the Zstandard data compression mechanism [13]. Zstandard is one of the eight compressors, along with GZIP [16], that are part of the IANA HTTP Content Coding Registry [42]. LZ4 is also the recommended data compression format for Cap'n Proto [69] bit-strings [43].

LZMA is an open-source compressor developed as part of the 7-Zip project [44]. LZMA offers high compression ratios as observed by [45] and [23]. LZMA is a dictionary-based compressor based on LZ77 [76] with support for dictionaries of up to 4 GB in size. As a result, LZMA can detect redundancy across large portions of the input data. In comparison to GZIP [16], [23] found LZMA to be space-efficient when taking large files as input. Applying LZMA on their 50 GB and a 440 GB collection of web pages resulted in 4.85% and 6.15% compression ratios compared to 20.35% and 18.69% compression ratios in the case of GZIP. LZMA support is implemented in the Opera web browser [45]. Official builds of the Firefox web browser do not include LZMA support. However, there exist non-official patches [46] that can be used to produce a build from source that includes LZMA support.

These data compression formats support multiple compression levels. We are interested in examining the impact of data compression in the best possible case, so we choose the highest recommended compression level supported by each format. The implementations, versions and compression levels used for this benchmark are listed in Table 9.

[39] `https://gnu.org`

[40] `https://ncompress.sourceforge.io`

[41] `https://github.com/Cyan4973`

[42] `https://www.iana.org/assignments/http-parameters/http-parameters.xhtml#content-coding`

[43] `https://capnproto.org/encoding.html#compression`

[44] `https://www.7-zip.org`

[45] `https://blogs.opera.com/desktop/changelog-for-31/`

[46] `https://wiki.mozilla.org/LZMA2_Compression`

Table 9: The selection of lossless data compression formats.

Format	Implementation	Compression Level	License
GZIP	Apple `gzip` 321.40.3 (based on FreeBSD `gzip` 20150113)	9 [47]	2-Clause BSD
LZ4	`lz4` command-line tool v1.9.3	9 [48]	Mixed 2-Clause BSD and GPLv2
LZMA	`xz` (XZ Utils) 5.2.5 with `liblzma` 5.2.5	9 [49]	Mixed Public Domain and GNU LGPLv2.1+

5.6 System Specification

The implementations of the selected serialization specifications were executed on a MacBook Pro 13" Dual-Core Intel Core i5 2.9 GHz with 2 cores and 16 GB of memory (model identifier `MacBookPro12,1`) running macOS Big Sur 11.2.3, Xcode 12.4 (12D4e), clang 1200.0.32.29, GNU Make 3.81, Matplotlib 3.4.2, Awk version 20200816, Python 3.9.2, Node.js 15.11.0, and Clojure 1.10.2.796.

6 Benchmark

In this section, we present bechmark results for 27 JSON [17] document examples as introduced in Table 4 and Table 5. We wrote schema definitions for each of the JSON documents for each of the 8 schema-driven binary serialization specifications introduced in Table 6 using their respective interface definition languages as exemplified in Figure 13.

In [71], we discussed that schema-driven serialization specifications typically define custom schema languages that are not usable by any other schema-driven serialization specifications rather than relying on standardized schema languages such as JSON Schema [75]. The selection of schema-driven binary serialization specifications implement schema languages that are considered high-level as they abstractly describe data structures and depend on the serialization specification implementation to provide the corresponding encoding rules. While schema definitions are typically written by hand, sometimes schemas are auto-generated from other schema languages [50] or inferred from the data [28] [38] [2] [15] [73] [19] [12] [64].

[50]`https://github.com/okdistribute/jsonschema-protobuf`

Apache Avro IDL - ECMAScript Module Loader Definition

```
{
  "namespace": "schema.avro",
  "type": "record",
  "name": "benchmark",
  "fields": [
    { "name": "cjs", "type": "boolean" },
    {
      "name": "mainFields",
      "type": {
        "type": "array",
        "items": "string"
      }
    },
    {
      "name": "mode",
      "type": {
        "type": "enum",
        "name": "mode",
        "symbols": [ "auto", "all", "strict" ]
      }
    },
    { "name": "force", "type": "boolean" },
    { "name": "cache", "type": "boolean" },
    { "name": "sourceMap", "type": "boolean" }
  ]
}
```

ASN.1 - Entry Point Regulation Manifest

```
GeneratedSchema DEFINITIONS AUTOMATIC TAGS ::= BEGIN
Rule ::= SEQUENCE {
    path UTF8String OPTIONAL,
    regex UTF8String OPTIONAL,
    types SEQUENCE OF UTF8String,
    allowData BOOLEAN
}
Main ::= SEQUENCE {
    site UTF8String,
    maxAge INTEGER,
    reportUrl UTF8String,
    defaultNavBehavior UTF8String,
    defaultResBehavior UTF8String,
    rules SEQUENCE OF Rule
}
END
```

Protocol Buffers IDL - CircleCI Definition (Blank)

```
syntax = "proto3";

message Main {
  float version = 1;
}
```

Figure 13: Examples of schema definitions written for 3 different JSON documents using 3 different interface definition languages: an Apache Avro Interface Definition Language (IDL) [27] schema definition for the JSON document presented in subsection 6.21 (left), an ASN.1 [56] schema definition for the JSON document presented in subsection 6.17 (top right) and a Protocol Buffers Interface Definition Language (IDL) [31] schema definition for the JSON document presented in subsection 6.3 (bottom right).

6.1 JSON-e Templating Engine Sort Example

JSON-e [51] is an open-source JSON-based templating engine created by Mozilla as part of the TaskCluster [52] project, the open-source task execution framework that supports Mozilla's continuous integration and release processes. In Figure 14, we demonstrate a **Tier 1 minified $<$ 100 bytes numeric redundant flat** (Tier 1 NRF from Table 2) JSON document that consists of an example JSON-e template definition to sort an array of numbers.

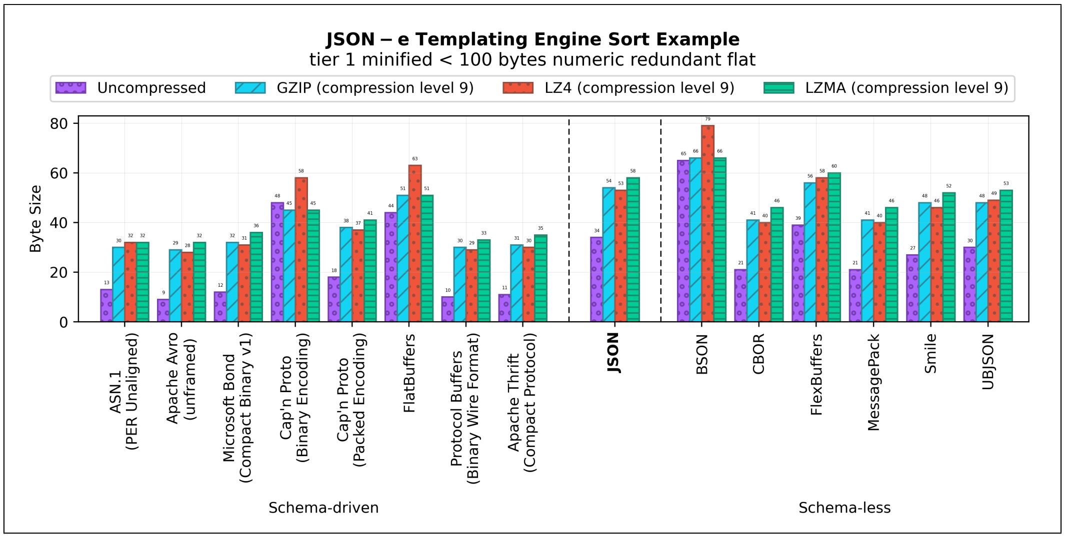

Figure 14: The benchmark results for the JSON-e Templating Engine Sort Example test case listed in Table 4 and Table 5.

The smallest bit-string is produced by Apache Avro [27] (9 bytes), closely followed by Protocol Buffers [31] (10 bytes) and Apache Thrift [60] (11 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by BSON [42] (65 bytes), followed by Cap'n Proto Binary Encoding [69] (48 bytes) and FlatBuffers [66] (44 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are schema-driven and pointer-based [71]. In comparison to JSON [17] (34 bytes), binary serialization achieves a **3.7x** size reduction in the best case for this input document. However, 4 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: Cap'n Proto Binary Encoding [69], FlatBuffers [66], BSON [42] and FlexBuffers [67]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 1 NRF document, the best performing schema-driven serialization specification achieves a **2.3x** size reduction compared to the best performing schema-less serialization specification: CBOR [5] and MessagePack [29] (21 bytes). As shown in Table 10, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. Additionally, as highlighted by the *range* and *standard deviation*, uncompressed schema-less specifications exhibit higher size reduction variability given that BSON [42] produces a notably large bit-string. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification achieves a **2x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (18 bytes).

The compression formats listed in subsection 5.5 result in positive gains for the bit-string produced by Cap'n Proto Binary Encoding [69]. The best performing uncompressed binary serialization

[51] `https://github.com/taskcluster/json-e`

[52] `https://taskcluster.net`

specification achieves a **5.8x** size reduction compared to the best performing compression format for JSON: LZ4 (53 bytes).

Table 10: A byte-size statistical analysis of the benchmark results shown in Figure 14 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	20.6	12.5	39	14.9	33.8	28.5	44	15.2
GZIP (compression level 9)	35.8	31.5	22	7.7	50	48	25	8.8
LZ4 (compression level 9)	38.5	31.5	35	13.0	52	47.5	39	13.5
LZMA (compression level 9)	38.1	35.5	19	6.5	53.8	52.5	20	7.2

Table 11: The benchmark raw data results and schemas for the plot in Figure 14.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	13	30	32	32
Apache Avro (unframed)	`schema.json`	9	29	28	32
Microsoft Bond (Compact Binary v1)	`schema.bond`	12	32	31	36
Cap'n Proto (Binary Encoding)	`schema.capnp`	48	45	58	45
Cap'n Proto (Packed Encoding)	`schema.capnp`	18	38	37	41
FlatBuffers	`schema.fbs`	44	51	63	51
Protocol Buffers (Binary Wire Format)	`schema.proto`	10	30	29	33
Apache Thrift (Compact Protocol)	`schema.thrift`	11	31	30	35
JSON	-	34	54	53	58
BSON	-	65	66	79	66
CBOR	-	21	41	40	46
FlexBuffers	-	39	56	58	60
MessagePack	-	21	41	40	46
Smile	-	27	48	46	52
UBJSON	-	30	48	49	53

In Figure 15, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers of uncompressed schema-driven binary serialization specifications. However, the inter-quartile range of both both uncompressed schema-driven and schema-less binary serialization specifications is similar. Additionally, their respective quartiles overlap.

In terms of compression, GZIP and LZ4 result in the lower medians for schema-driven binary serialization specifications while LZ4 results in the lower median for schema-less binary serialization specifications. However, compression is not space-efficient in terms of the median for both schema-driven and schema-less binary serialization specifications. Additionally, the use of LZ4 for both schema-driven binary serialization specifications and schema-less binary serialization specifications exhibits upper outliers. While compression does not contribute to space-efficiency, it reduces the range between the upper and lower whiskers and inter-quartile range for schema-driven binary serialization specifications and it reduces the inter-quartile range for schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and

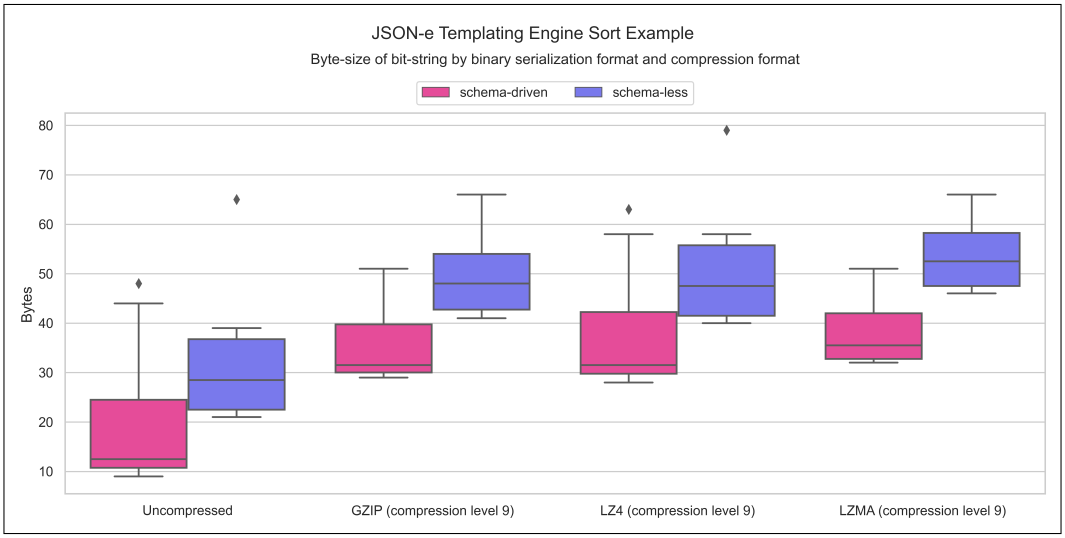

Figure 15: Box plot of the statistical results in Table 10.

lower whiskers for schema-driven binary serialization specifications is LZMA, the compression formats with the smaller inter-quartile range for schema-driven binary serialization specifications are GZIP and LZMA, the compression format with the smaller range between the upper and lower whiskers for schema-less binary serialization specifications is LZ4, and the compression formats with the smaller inter-quartile range for schema-less binary serialization specifications are GZIP and LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that compression does not contribute to space-efficiency in comparison to both uncompressed schema-driven and schema-less binary serialization specifications.

6.2 JSON-e Templating Engine Reverse Sort Example

JSON-e [53] is an open-source JSON-based templating engine created by Mozilla as part of the TaskCluster [54] project, the open-source task execution framework that supports Mozilla's continuous integration and release processes. In Figure 16, we demonstrate a **Tier 1 minified $<$ 100 bytes numeric redundant nested** (Tier 1 NRN from Table 2) JSON document that consists of an example JSON-e template definition to sort and reverse an array of numbers.

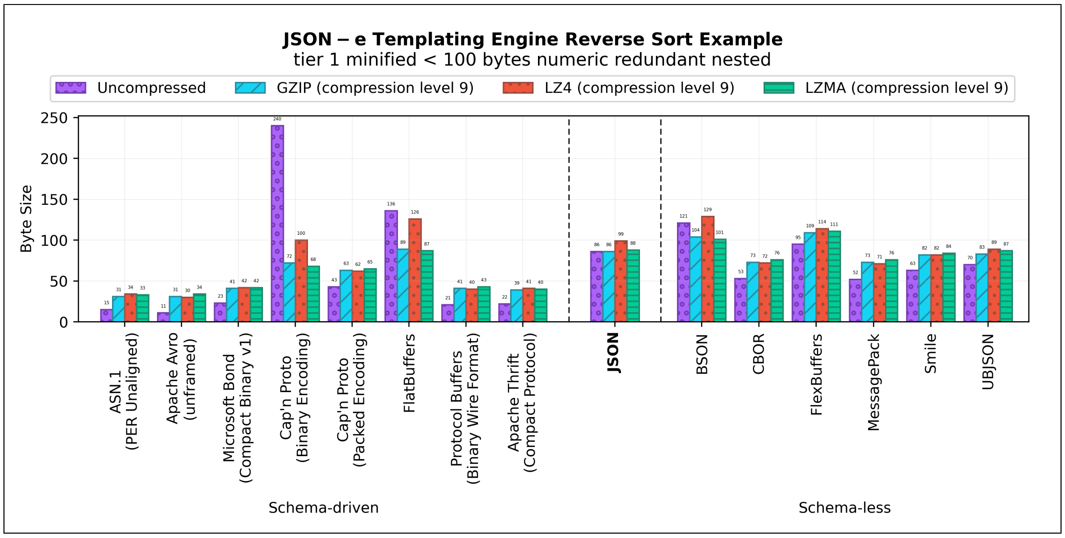

Figure 16: The benchmark results for the JSON-e Templating Engine Reverse Sort Example test case listed in Table 4 and Table 5.

The smallest bit-string is produced by Apache Avro [27] (11 bytes), followed by ASN.1 PER Unaligned [57] (15 bytes) and Protocol Buffers [31] (21 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by Cap'n Proto Binary Encoding [69] (240 bytes), followed by FlatBuffers [66] (136 bytes) and BSON [42] (121 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are schema-driven and pointer-based [71]. In comparison to JSON [17] (86 bytes), binary serialization achieves a **7.8x** size reduction in the best case for this input document. However, 4 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: Cap'n Proto Binary Encoding [69], FlatBuffers [66], BSON [42] and FlexBuffers [67]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 1 NRN document, the best performing schema-driven serialization specification achieves a **4.7x** size reduction compared to the best performing schema-less serialization specification: CBOR [5] and MessagePack [29] (52 bytes). As shown in Table 12, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification achieves a **3.9x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (43 bytes).

The compression formats listed in subsection 5.5 result in positive gains for the bit-strings produced by Cap'n Proto Binary Encoding [69], FlatBuffers [66] and BSON [42]. The best performing

[53] `https://github.com/taskcluster/json-e`

[54] `https://taskcluster.net`

uncompressed binary serialization specification achieves a **7.8x** size reduction compared to the best performing compression format for JSON: GZIP [16] (86 bytes).

Table 12: A byte-size statistical analysis of the benchmark results shown in Figure 16 divided by schema-driven and schema-less specifications.

Category	Schema-driven				Schema-less			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	63.9	22.5	229	76.7	75.7	66.5	69	24.8
GZIP (compression level 9)	50.9	41	58	19.9	87.3	82.5	36	14.2
LZ4 (compression level 9)	59.4	41.5	96	32.8	92.8	85.5	58	21.6
LZMA (compression level 9)	51.5	42.5	54	18.2	89.2	85.5	35	12.9

Table 13: The benchmark raw data results and schemas for the plot in Figure 16.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	15	31	34	33
Apache Avro (unframed)	`schema.json`	11	31	30	34
Microsoft Bond (Compact Binary v1)	`schema.bond`	23	41	42	42
Cap'n Proto (Binary Encoding)	`schema.capnp`	240	72	100	68
Cap'n Proto (Packed Encoding)	`schema.capnp`	43	63	62	65
FlatBuffers	`schema.fbs`	136	89	126	87
Protocol Buffers (Binary Wire Format)	`schema.proto`	21	41	40	43
Apache Thrift (Compact Protocol)	`schema.thrift`	22	39	41	40
JSON	-	86	86	99	88
BSON	-	121	104	129	101
CBOR	-	53	73	72	76
FlexBuffers	-	95	109	114	111
MessagePack	-	52	73	71	76
Smile	-	63	82	82	84
UBJSON	-	70	83	89	87

In Figure 17, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications. However, their respective quartiles overlap.

In terms of compression, GZIP, LZ4 and LZMA resuls in similar medians for schema-driven binary serialization specifications while GZIP results in the lower median for schema-less binary serialization specifications. However, compression is not space-efficient in terms of the median for both schema-driven and schema-less binary serialization specifications. Additionally, the use of LZ4 for schema-driven binary serialization specifications exhibits upper outliers. While compression does not contribute to space-efficiency, it reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression formats with the smaller range between the upper and lower whiskers for schema-driven binary serialization specifications are GZIP and LZMA, the compression formats with the smaller inter-quartile range for schema-driven binary serialization specifications are GZIP and

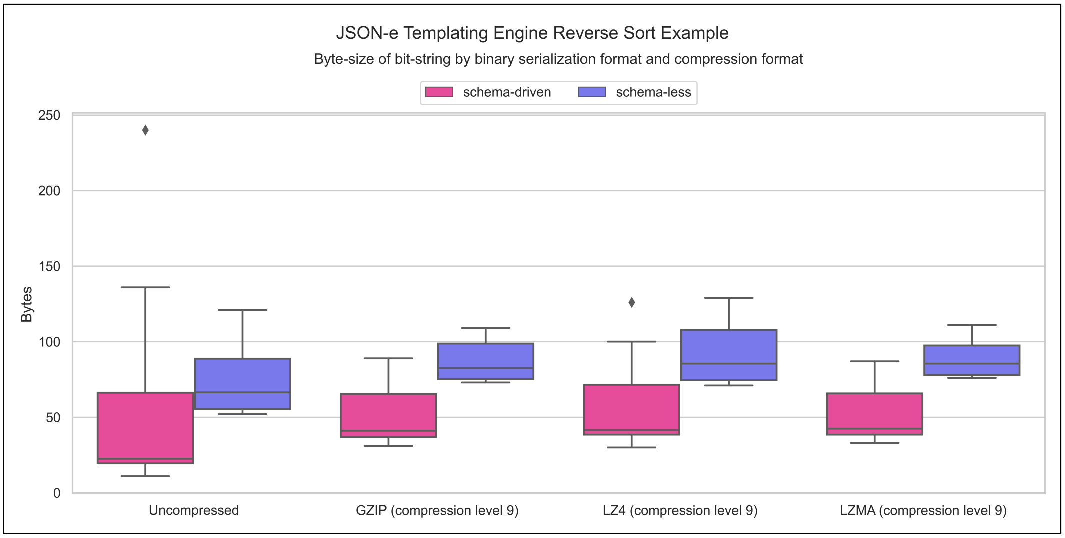

Figure 17: Box plot of the statistical results in Table 12.

LZMA, the compression formats with the smaller range between the upper and lower whiskers for schema-less binary serialization specifications are GZIP and LZMA, and the compression format with the smaller inter-quartile range for schema-less binary serialization specifications is LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that compression does not contribute to space-efficiency in comparison to both uncompressed schema-driven and schema-less binary serialization specifications.

6.3 CircleCI Definition (Blank)

CircleCI [55] is a commercial cloud-provider of continuous integration and deployment pipelines used by a wide range of companies in the software development industry such as Facebook, Spotify, and Heroku [56]. In Figure 18, we demonstrate a **Tier 1 minified $<$ 100 bytes numeric non-redundant flat** (Tier 1 NNF from Table 2) JSON document that represents a simple pipeline configuration file for CircleCI that declares the desired CircleCI version without defining any workflows.

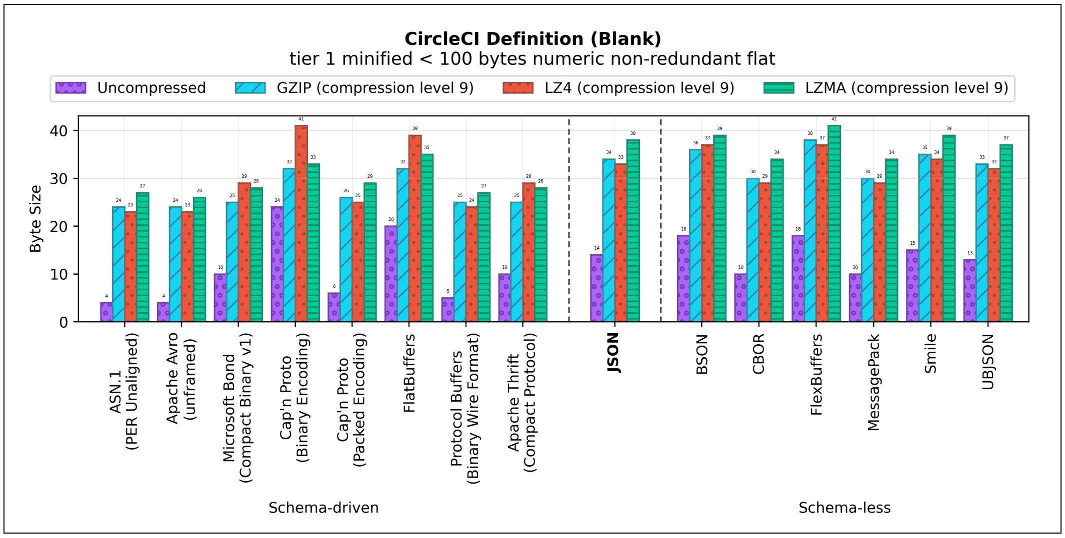

Figure 18: The benchmark results for the CircleCI Definition (Blank) test case listed in Table 4 and Table 5.

The smallest bit-string is produced by both ASN.1 PER Unaligned [57] and Apache Avro [27] (4 bytes), closely followed by Protocol Buffers [31] (5 bytes) and Cap'n Proto Packed Encoding [69] (6 bytes). These serialization specifications are schema-driven and with the exception of Cap'n Proto Packed Encoding, which occupies the fourth place, they are also sequential [71]. Conversely, the largest bit-string is produced by Cap'n Proto Binary Encoding [69] (24 bytes), followed by FlatBuffers [66] (20 bytes) and both BSON [42] and FlexBuffers [67] (18 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are pointer-based [71]. In comparison to JSON [17] (14 bytes), binary serialization achieves a **3.5x** size reduction in the best case for this input document. However, 5 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: Cap'n Proto Binary Encoding [69], FlatBuffers [66], BSON [42], FlexBuffers [67] and Smile [55]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 1 NNF document, the best performing schema-driven serialization specification achieves a **2.5x** size reduction compared to the best performing schema-less serialization specification: CBOR [5] and MessagePack [29] (10 bytes). As shown in Table 14, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the schema language capabilities and implemented size optimizations. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification only achieves a **1.5x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (6 bytes).

The compression formats listed in subsection 5.5 do not result in positive gains for any case due to the overhead of encoding the dictionary data structures and the low redundancy of the input data.

[55] `https://circleci.com`
[56] `https://circleci.com/customers/`

The best performing uncompressed binary serialization specification achieves a **8.2x** size reduction compared to the best performing compression format for JSON: LZ4 (33 bytes).

Table 14: A byte-size statistical analysis of the benchmark results shown in Figure 18 divided by schema-driven and schema-less specifications.

Category	Schema-driven				Schema-less			
	Average	Median	Range	Std.dev	Average	Median	Range	Std.dev
Uncompressed	10.4	8	20	7.1	14	14	8	3.3
GZIP (compression level 9)	26.6	25	8	3.2	33.7	34	8	3.0
LZ4 (compression level 9)	29.1	27	18	6.7	33	33	8	3.3
LZMA (compression level 9)	29.1	28	9	3.0	37.3	38	7	2.6

Table 15: The benchmark raw data results and schemas for the plot in Figure 18.

Serialization Format	Schema	Uncompressed	GZIP	LZ4	LZMA
ASN.1 (PER Unaligned)	`schema.asn`	4	24	23	27
Apache Avro (unframed)	`schema.json`	4	24	23	26
Microsoft Bond (Compact Binary v1)	`schema.bond`	10	25	29	28
Cap'n Proto (Binary Encoding)	`schema.capnp`	24	32	41	33
Cap'n Proto (Packed Encoding)	`schema.capnp`	6	26	25	29
FlatBuffers	`schema.fbs`	20	32	39	35
Protocol Buffers (Binary Wire Format)	`schema.proto`	5	25	24	27
Apache Thrift (Compact Protocol)	`schema.thrift`	10	25	29	28
JSON	-	14	34	33	38
BSON	-	18	36	37	39
CBOR	-	10	30	29	34
FlexBuffers	-	18	38	37	41
MessagePack	-	10	30	29	34
Smile	-	15	35	34	39
UBJSON	-	13	33	32	37

In Figure 19, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications. However, their respective quartiles overlap.

In terms of compression, GZIP results in the lower median for schema-driven binary serialization specifications while LZ4 results in the lower median for schema-less binary serialization specifications. However, compression is not space-efficient in terms of the median for both schema-driven and schema-less binary serialization specifications. Additionally, the use of GZIP and LZMA for schema-driven binary serialization specifications exhibit upper outliers. While compression does not contribute to space-efficiency, it reduces the range between the upper and lower whiskers and inter-quartile range for schema-driven binary serialization specifications and it reduces the inter-quartile range for schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers for schema-driven binary serialization specifications is GZIP, the compression formats with the smaller inter-quartile range for schema-driven

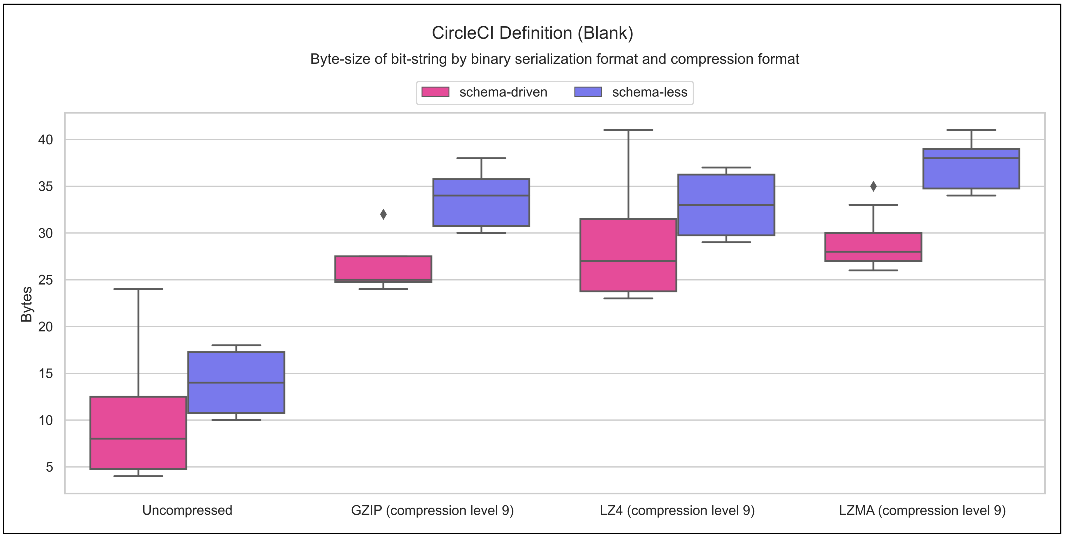

Figure 19: Box plot of the statistical results in Table 14.

binary serialization specifications are GZIP and LZMA, and the compression format with the smaller inter-quartile range for schema-less binary serialization specifications is LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that compression does not contribute to space-efficiency in comparison to both uncompressed schema-driven and schema-less binary serialization specifications.

6.4 CircleCI Matrix Definition

CircleCI [57] is a commercial cloud-provider of continuous integration and deployment pipelines used by a wide range of companies in the software development industry such as Facebook, Spotify, and Heroku [58]. In Figure 20, we demonstrate a **Tier 1 minified $<$ 100 bytes numeric non-redundant nested** (Tier 1 NNN from Table 2) JSON document that represents a pipeline configuration file for CircleCI that declares the desired CircleCI version and defines a workflow that contains a single blank matrix-based job.

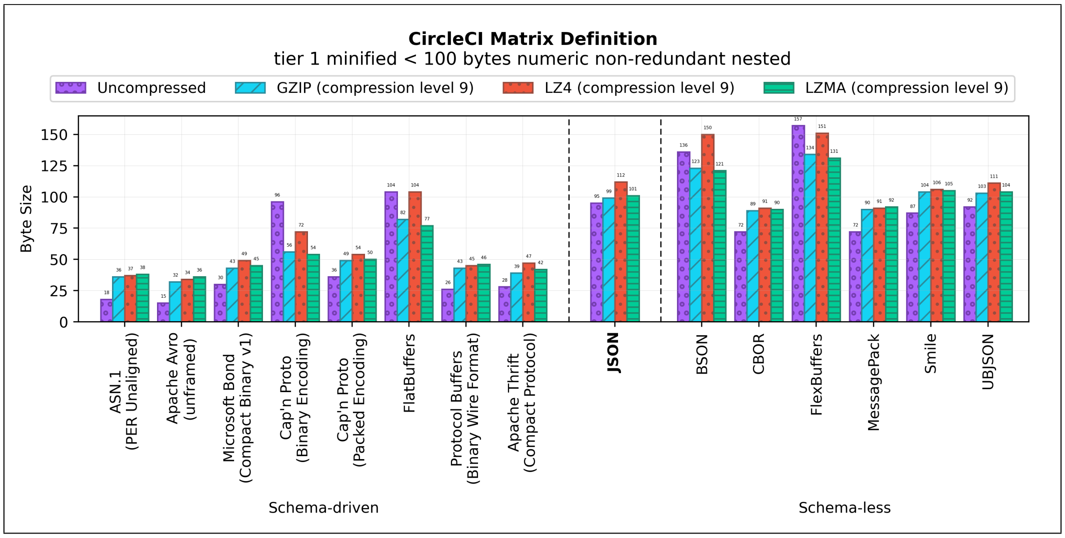

Figure 20: The benchmark results for the CircleCI Matrix Definition test case listed in Table 4 and Table 5.

The smallest bit-string is produced by Apache Avro [27] (15 bytes), followed by ASN.1 PER Unaligned [57] (18 bytes) and Protocol Buffers [31] (26 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by FlexBuffers [67] (157 bytes), BSON [42] (136 bytes) and FlatBuffers [66] (104 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are pointer-based [71]. In comparison to JSON [17] (95 bytes), binary serialization achieves a **6.3x** size reduction in the best case for this input document. However, 4 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: Cap'n Proto Binary Encoding [69], FlatBuffers [66], BSON [42] and FlexBuffers [67]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 1 NNN document, the best performing schema-driven serialization specification achieves a **4.8x** size reduction compared to the best performing schema-less serialization specification: CBOR [5] and MessagePack [29] (72 bytes). As shown in Table 16, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the schema language capabilities and implemented size optimizations. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification achieves a **2.4x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (36 bytes).

The compression formats listed in subsection 5.5 result in positive gains for the bit-strings produced by Cap'n Proto Binary Encoding [69], FlatBuffers [66], BSON [42] and FlexBuffers [67]. The best

[57] `https://circleci.com`

[58] `https://circleci.com/customers/`

performing uncompressed binary serialization specification achieves a **6.6x** size reduction compared to the best performing compression format for JSON: GZIP [16] (99 bytes).

Table 16: A byte-size statistical analysis of the benchmark results shown in Figure 20 divided by schema-driven and schema-less specifications.

Category	Schema-driven				Schema-less			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	44.1	29	89	32.9	102.7	89.5	85	32.4
GZIP (compression level 9)	47.5	43	50	14.8	107.2	103.5	45	16.4
LZ4 (compression level 9)	55.3	48	70	21.4	116.7	108.5	60	25.0
LZMA (compression level 9)	48.5	45.5	41	12.1	107.2	104.5	41	14.7

Table 17: The benchmark raw data results and schemas for the plot in Figure 20.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	18	36	37	38
Apache Avro (unframed)	`schema.json`	15	32	34	36
Microsoft Bond (Compact Binary v1)	`schema.bond`	30	43	49	45
Cap'n Proto (Binary Encoding)	`schema.capnp`	96	56	72	54
Cap'n Proto (Packed Encoding)	`schema.capnp`	36	49	54	50
FlatBuffers	`schema.fbs`	104	82	104	77
Protocol Buffers (Binary Wire Format)	`schema.proto`	26	43	45	46
Apache Thrift (Compact Protocol)	`schema.thrift`	28	39	47	42
JSON	-	95	99	112	101
BSON	-	136	123	150	121
CBOR	-	72	89	91	90
FlexBuffers	-	157	134	151	131
MessagePack	-	72	90	91	92
Smile	-	87	104	106	105
UBJSON	-	92	103	111	104

In Figure 21, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications.

In terms of compression, LZMA results in the lower median for schema-driven binary serialization specifications while GZIP results in the lower median for schema-less binary serialization specifications. However, compression is not space-efficient in terms of the median for both schema-driven and schema-less binary serialization specifications. Additionally, the use of GZIP, LZ4 and LZMA for schema-driven binary serialization specifications exhibit upper outliers. While compression does not contribute to space-efficiency, it reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA, the compression formats with the smaller range between the upper and lower whiskers for schema-less

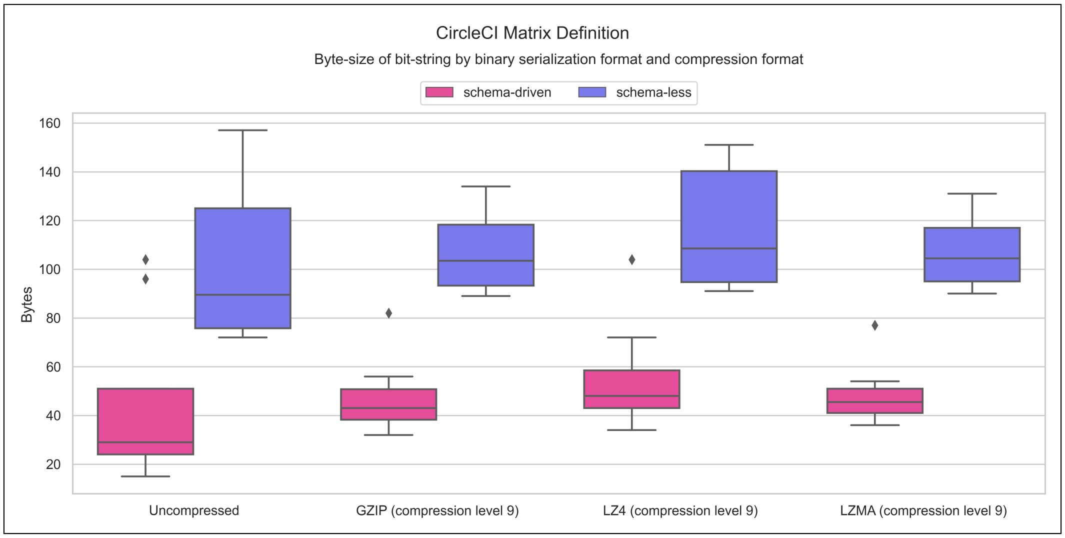

Figure 21: Box plot of the statistical results in Table 16.

binary serialization specifications are GZIP and LZMA, and the compression format with the smaller inter-quartile range for schema-less binary serialization specifications is LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that compression does not contribute to space-efficiency in comparison to both uncompressed schema-driven and schema-less binary serialization specifications.

6.5 Grunt.js Clean Task Definition

Grunt.js [59] is an open-source task runner for the JavaScript [18] programming language used by a wide range of companies in the software development industry such as Twitter, Adobe, and Mozilla [60]. In Figure 22, we demonstrate a **Tier 1 minified < 100 bytes textual redundant flat** (Tier 1 TRF from Table 2) JSON document that consists of an example configuration for a built-in plugin to clear files and folders called `grunt-contrib-clean` [61].

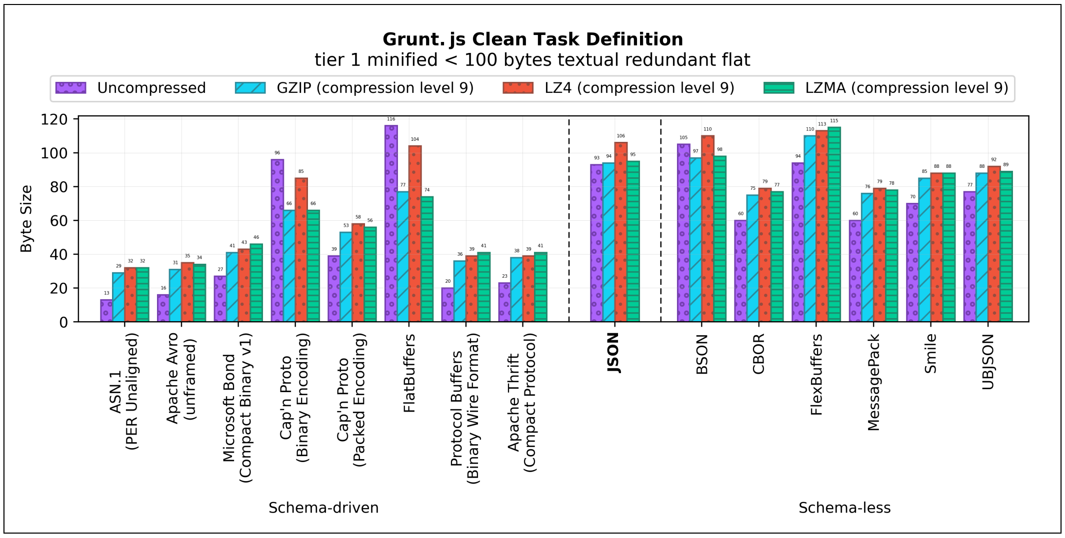

Figure 22: The benchmark results for the Grunt.js Clean Task Definition test case listed in Table 4 and Table 5.

The smallest bit-string is produced by ASN.1 PER Unaligned [57] (13 bytes), followed by Apache Avro [27] (16 bytes) and Protocol Buffers [31] (20 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by FlatBuffers [66] (116 bytes), followed by BSON [42] (105 bytes) and Cap'n Proto Binary Encoding [69] (96 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are schema-driven and pointer-based [71]. In comparison to JSON [17] (93 bytes), binary serialization achieves a **7.1x** size reduction in the best case for this input document. However, 4 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: Cap'n Proto Binary Encoding [69], FlatBuffers [66], BSON [42] and FlexBuffers [67]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 1 TRF document, the best performing schema-driven serialization specification achieves a **4.6x** size reduction compared to the best performing schema-less serialization specification: CBOR [5] and MessagePack [29] (60 bytes). As shown in Table 18, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification achieves a **3x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (39 bytes).

[59] `https://gruntjs.com`
[60] `https://gruntjs.com/who-uses-grunt`
[61] `https://github.com/gruntjs/grunt-contrib-clean`

The compression formats listed in subsection 5.5 result in positive gains for the bit-strings produced by Cap'n Proto Binary Encoding [69], FlatBuffers [66] and BSON [42]. The best performing uncompressed binary serialization specification achieves a **7.2x** size reduction compared to the best performing compression format for JSON: GZIP [16] (94 bytes).

Table 18: A byte-size statistical analysis of the benchmark results shown in Figure 22 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	43.8	25	103	37.0	77.7	73.5	45	16.8
GZIP (compression level 9)	46.4	39.5	48	16.2	88.5	86.5	35	12.1
LZ4 (compression level 9)	54.4	41	72	24.7	93.5	90	34	13.6
LZMA (compression level 9)	48.8	43.5	42	14.2	90.8	88.5	38	12.9

Table 19: The benchmark raw data results and schemas for the plot in Figure 22.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	13	29	32	32
Apache Avro (unframed)	`schema.json`	16	31	35	34
Microsoft Bond (Compact Binary v1)	`schema.bond`	27	41	43	46
Cap'n Proto (Binary Encoding)	`schema.capnp`	96	66	85	66
Cap'n Proto (Packed Encoding)	`schema.capnp`	39	53	58	56
FlatBuffers	`schema.fbs`	116	77	104	74
Protocol Buffers (Binary Wire Format)	`schema.proto`	20	36	39	41
Apache Thrift (Compact Protocol)	`schema.thrift`	23	38	39	41
JSON	-	93	94	106	95
BSON	-	105	97	110	98
CBOR	-	60	75	79	77
FlexBuffers	-	94	110	113	115
MessagePack	-	60	76	79	78
Smile	-	70	85	88	88
UBJSON	-	77	88	92	89

In Figure 23, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications.

In terms of compression, GZIP results in the lower median for both schema-driven and schema-less binary serialization specifications. However, compression is not space-efficient in terms of the median for both schema-driven and schema-less binary serialization specifications. While compression does not contribute to space-efficiency, it reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers for schema-driven binary serialization specifications is LZMA, the compression formats with the smaller inter-quartile range for schema-driven binary serialization specifications are GZIP and LZMA, the

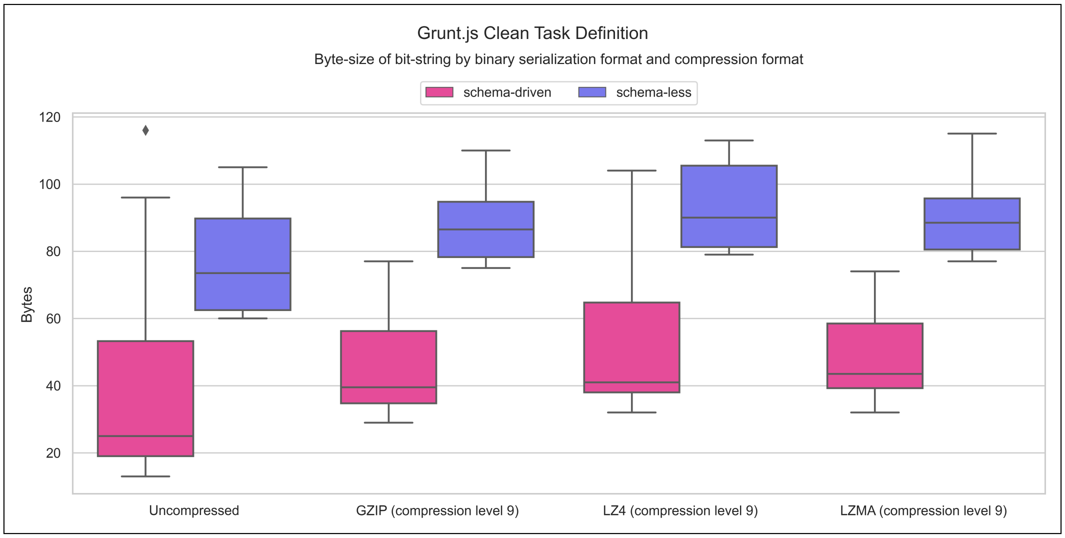

Figure 23: Box plot of the statistical results in Table 18.

compression formats with the smaller range between the upper and lower whiskers for schema-less binary serialization specifications are GZIP and LZ4, and the compression formats with the smaller inter-quartile range for schema-less binary serialization specifications are GZIP and LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that compression does not contribute to space-efficiency in comparison to both uncompressed schema-driven and schema-less binary serialization specifications.

6.6 CommitLint Configuration

CommitLint [62] is an open-source command-line tool to enforce version-control commit conventions in software engineering projects. CommitLint is a community effort under the Conventional Changelog [63] organization formed by employees from companies including GitHub [64] and Google [65]. In Figure 24, we demonstrate a **Tier 1 minified $<$ 100 bytes textual redundant nested** (Tier 1 TRN from Table 2) JSON document that represents a CommitLint configuration file which declares that the subject and the scope of any commit must be written in lower-case form.

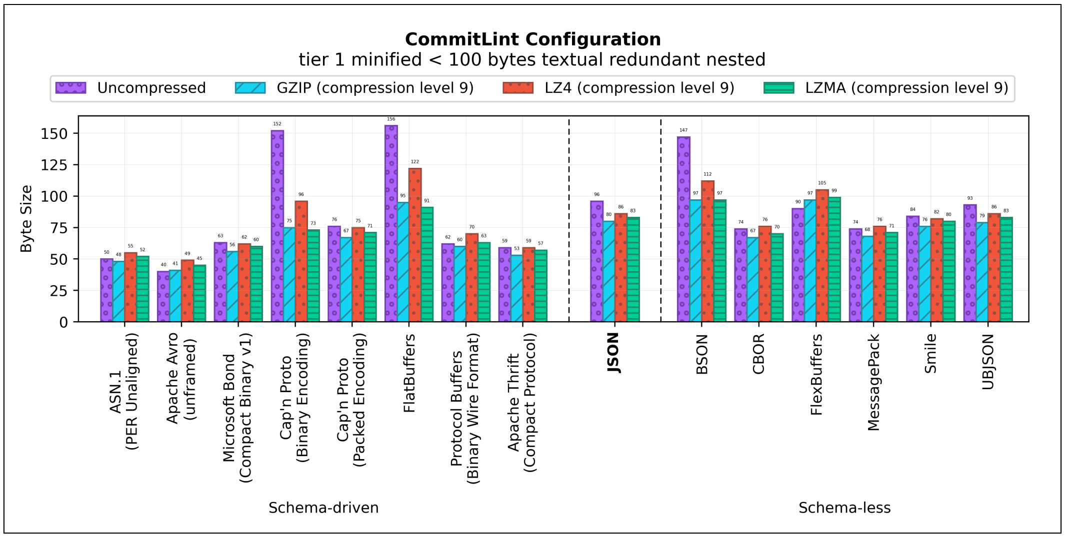

Figure 24: The benchmark results for the CommitLint Configuration test case listed in Table 4 and Table 5.

The smallest bit-string is produced by Apache Avro [27] (40 bytes), followed by ASN.1 PER Unaligned [57] (50 bytes) and Apache Thrift [60] (59 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by FlatBuffers [66] (156 bytes), Cap'n Proto Binary Encoding [69] (152 bytes) and BSON [42] (147 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are pointer-based [71]. In comparison to JSON [17] (96 bytes), binary serialization achieves a **2.4x** size reduction in the best case for this input document. However, 3 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: Cap'n Proto Binary Encoding [69], FlatBuffers [66] and BSON [42]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 1 TRN document, the best performing schema-driven serialization specification only achieves a **1.8x** size reduction compared to the best performing schema-less serialization specification: CBOR [5] and MessagePack [29] (74 bytes). As shown in Table 20, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69], Cap'n Proto Packed Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing se-

[62] `https://commitlint.js.org/#/`

[63] `https://github.com/conventional-changelog`

[64] `https://github.com/zeke`

[65] `https://github.com/bcoe`

quential serialization specification achieves a **1.9x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (76 bytes).

The compression formats listed in subsection 5.5 result in positive gains for all the bit-strings except the ones produced by Apache Avro [27] and FlexBuffers [67]. The best performing uncompressed binary serialization specification achieves a **2x** size reduction compared to the best performing compression format for JSON: GZIP [16] (80 bytes).

Table 20: A byte-size statistical analysis of the benchmark results shown in Figure 24 divided by schema-driven and schema-less specifications.

Category	Schema-driven				Schema-less			
	Average	Median	Range	Std.dev	Average	Median	Range	Std.dev
Uncompressed	82.3	62.5	116	42.6	93.7	87	73	24.9
GZIP (compression level 9)	61.9	58	54	16.0	80.7	77.5	30	12.3
LZ4 (compression level 9)	73.5	66	73	22.8	89.5	84	36	14.0
LZMA (compression level 9)	64	61.5	46	13.4	83.3	81.5	29	11.4

Table 21: The benchmark raw data results and schemas for the plot in Figure 24.

Serialization Format	Schema	Uncompressed	GZIP	LZ4	LZMA
ASN.1 (PER Unaligned)	`schema.asn`	50	48	55	52
Apache Avro (unframed)	`schema.json`	40	41	49	45
Microsoft Bond (Compact Binary v1)	`schema.bond`	63	56	62	60
Cap'n Proto (Binary Encoding)	`schema.capnp`	152	75	96	73
Cap'n Proto (Packed Encoding)	`schema.capnp`	76	67	75	71
FlatBuffers	`schema.fbs`	156	95	122	91
Protocol Buffers (Binary Wire Format)	`schema.proto`	62	60	70	63
Apache Thrift (Compact Protocol)	`schema.thrift`	59	53	59	57
JSON	-	96	80	86	83
BSON	-	147	97	112	97
CBOR	-	74	67	76	70
FlexBuffers	-	90	97	105	99
MessagePack	-	74	68	76	71
Smile	-	84	76	82	80
UBJSON	-	93	79	86	83

In Figure 25, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications. However, their respective quartiles overlap.

In terms of compression, GZIP results in the lower median for both schema-driven and schema-less binary serialization specifications. Additionally, GZIP and LZMA are space-efficient in terms of the median in comparison to uncompressed schema-driven binary serialization specifications and GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to uncompressed schema-less binary serialization specifications. However, the use of GZIP and LZ4 for schema-driven

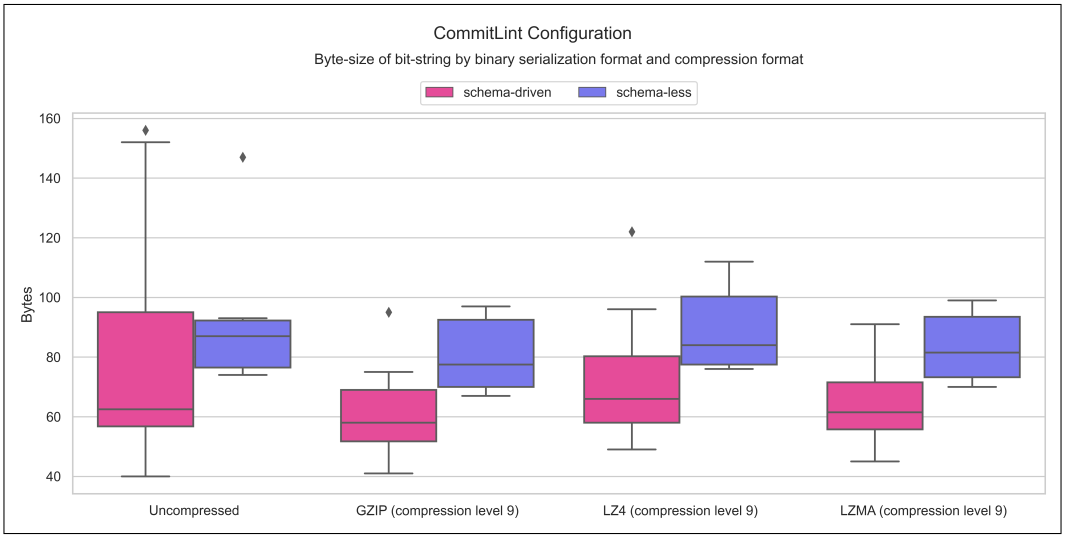

Figure 25: Box plot of the statistical results in Table 20.

binary serialization specifications exhibit upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for schema-driven binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers for schema-driven binary serialization specifications is GZIP, the compression format with the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA, and the compression formats with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-less binary serialization specifications are GZIP and LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications. GZIP and LZMA are space-efficient in comparison to uncompressed schema-driven binary serialization specifications and all the considered compression formats are space-efficient in comparison to uncompressed schema-less binary serialization specifications.

6.7 TSLint Linter Definition (Extends Only)

TSLint [66] is now an obsolete open-source linter for the TypeScript [67] programming language. TSLint was created by the Big Data analytics company Palantir [68] and was merged with the ESLint open-source JavaScript linter in 2019 [69]. In Figure 26, we demonstrate a **Tier 1 minified $<$ 100 bytes textual non-redundant flat** (Tier 1 TNF from Table 2) JSON document that consists of a basic TSLint configuration that only extends a set of existing TSLint configurations.

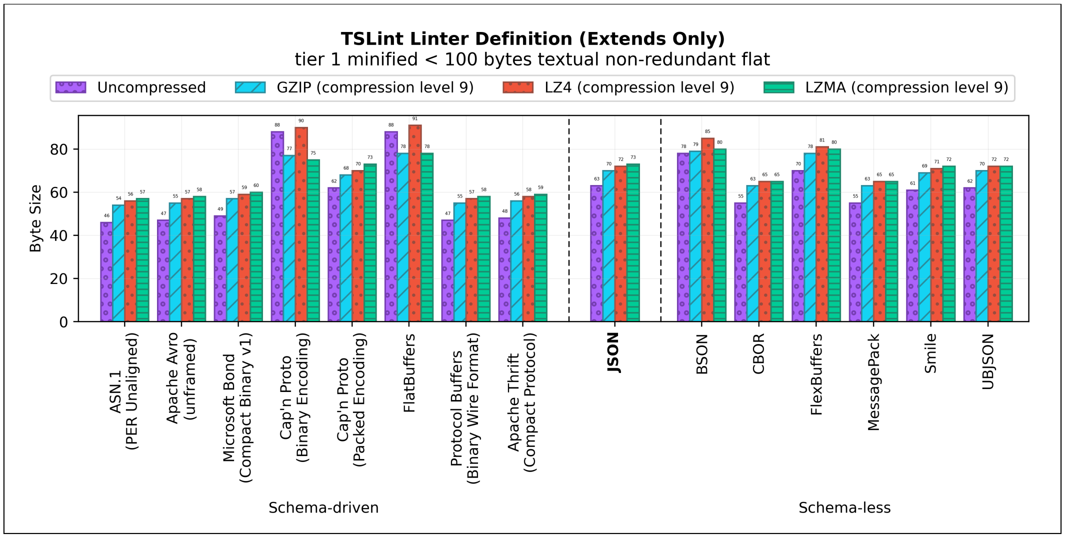

Figure 26: The benchmark results for the TSLint Linter Definition (Extends Only) test case listed in Table 4 and Table 5.

The smallest bit-string is produced by ASN.1 PER Unaligned [57] (46 bytes), closely followed by both Apache Avro [27] and Protocol Buffers [31] (47 bytes), and Apache Thrift [60] (48 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by both Cap'n Proto Binary Encoding [69] and FlatBuffers [66] (88 bytes), followed by BSON [42] (78 bytes) and FlexBuffers [67] (70 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are pointer-based [71]. In comparison to JSON [17] (63 bytes), binary serialization only achieves a **1.3x** size reduction in the best case for this input document. Additionally, 4 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: Cap'n Proto Binary Encoding [69], FlatBuffers [66], BSON [42] and FlexBuffers [67]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 1 TNF document, the best performing schema-driven serialization specification only achieves a **1.1x** size reduction compared to the best performing schema-less serialization specification: CBOR [5] and MessagePack [29] (55 bytes). As shown in Table 22, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69], Cap'n Proto Packed Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequen-

[66] `https://palantir.github.io/tslint`
[67] `https://www.typescriptlang.org`
[68] `https://www.palantir.com`
[69] `https://github.com/palantir/tslint/issues/4534`

tial serialization specification only achieves a **1.3x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (62 bytes).

The compression formats listed in subsection 5.5 result in positive gains for the bit-strings produced by Cap'n Proto Binary Encoding [69] and FlatBuffers [66]. The best performing uncompressed binary serialization specification achieves a **1.5x** size reduction compared to the best performing compression format for JSON: GZIP [16] (70 bytes).

Table 22: A byte-size statistical analysis of the benchmark results shown in Figure 26 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	59.4	48.5	42	17.2	63.5	61.5	23	8.2
GZIP (compression level 9)	62.5	56.5	24	9.6	70.3	69.5	16	6.4
LZ4 (compression level 9)	67.3	58.5	35	14.1	73.2	71.5	20	7.5
LZMA (compression level 9)	64.8	59.5	21	8.3	72.3	72	15	6.1

Table 23: The benchmark raw data results and schemas for the plot in Figure 26.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	46	54	56	57
Apache Avro (unframed)	`schema.json`	47	55	57	58
Microsoft Bond (Compact Binary v1)	`schema.bond`	49	57	59	60
Cap'n Proto (Binary Encoding)	`schema.capnp`	88	77	90	75
Cap'n Proto (Packed Encoding)	`schema.capnp`	62	68	70	73
FlatBuffers	`schema.fbs`	88	78	91	78
Protocol Buffers (Binary Wire Format)	`schema.proto`	47	55	57	58
Apache Thrift (Compact Protocol)	`schema.thrift`	48	56	58	59
JSON	-	63	70	72	73
BSON	-	78	79	85	80
CBOR	-	55	63	65	65
FlexBuffers	-	70	78	81	80
MessagePack	-	55	63	65	65
Smile	-	61	69	71	72
UBJSON	-	62	70	72	72

In Figure 27, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications. However, their respective quartiles overlap.

In terms of compression, GZIP results in the lower median for both schema-driven and schema-less binary serialization specifications. However, compression is not space-efficient in terms of the median for both schema-driven and schema-less binary serialization specifications. While compression does not contribute to space-efficiency, it reduces the range between the upper and lower whiskers and inter-quartile range for schema-driven binary serialization specifications. In

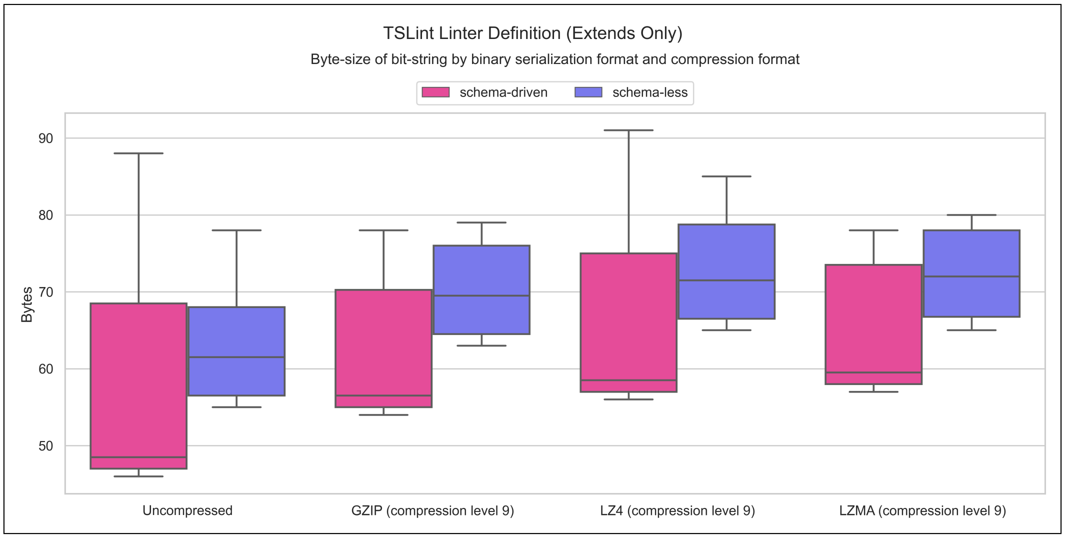

Figure 27: Box plot of the statistical results in Table 22.

particular, the compression format with the smaller range between the upper and lower whiskers for schema-driven binary serialization specifications is LZMA, the compression formats with the smaller inter-quartile range for schema-driven binary serialization specifications are GZIP and LZMA, and the compression format with the smaller range between the upper and lower whiskers for schema-less binary serialization specifications is LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that compression does not contribute to space-efficiency in comparison to both uncompressed schema-driven and schema-less binary serialization specifications.

6.8 ImageOptimizer Azure Webjob Configuration

Image Optimizer [70] is an Azure App Services WebJob [71] to compress website images used in the web development industry. In Figure 28, we demonstrate a **Tier 1 minified < 100 bytes textual non-redundant nested** (Tier 1 TNN from Table 2) JSON document that consists of an Image Optimizer configuration to perform lossy compression on images inside a particular folder.

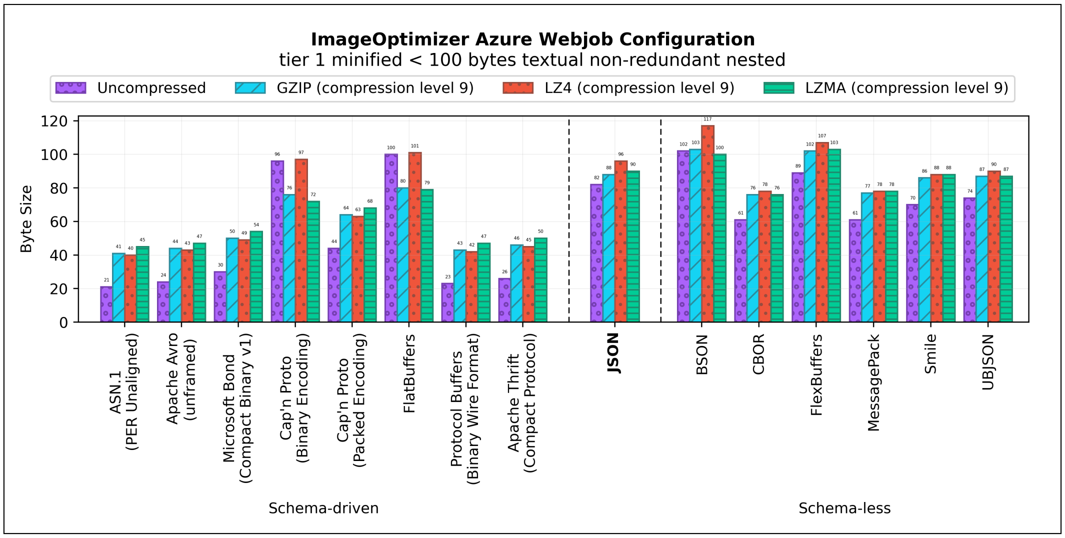

Figure 28: The benchmark results for the ImageOptimizer Azure Webjob Configuration test case listed in Table 4 and Table 5.

The smallest bit-string is produced by ASN.1 PER Unaligned [57] (21 bytes), closely followed by Protocol Buffers [31] (23 bytes) and Apache Avro [27] (24 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by BSON [42] (102 bytes), closely followed by FlatBuffers [66] (100 bytes) and Cap'n Proto Binary Encoding [69] (96 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are schema-driven and pointer-based [71]. In comparison to JSON [17] (82 bytes), binary serialization achieves a **3.9x** size reduction in the best case for this input document. However, 4 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: Cap'n Proto Binary Encoding [69], FlatBuffers [66], BSON [42] and FlexBuffers [67]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 1 TNN document, the best performing schema-driven serialization specification achieves a **2.9x** size reduction compared to the best performing schema-less serialization specification: CBOR [5] and MessagePack [29] (61 bytes). As shown in Table 24, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification achieves a **2x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (44 bytes).

The compression formats listed in subsection 5.5 result in positive gains for the bit-strings produced by Cap'n Proto Binary Encoding [69] and FlatBuffers [66]. The best performing uncompressed

[70] `https://github.com/madskristensen/ImageOptimizerWebJob`

[71] `https://docs.microsoft.com/en-us/azure/app-service/webjobs-create`

binary serialization specification achieves a **4.1x** size reduction compared to the best performing compression format for JSON: GZIP [16] (88 bytes).

Table 24: A byte-size statistical analysis of the benchmark results shown in Figure 28 divided by schema-driven and schema-less specifications.

Category	Schema-driven				Schema-less			
	Average	Median	Range	Std.dev	Average	Median	Range	Std.dev
Uncompressed	45.5	28	79	31.0	76.2	72	41	14.9
GZIP (compression level 9)	55.5	48	39	14.6	88.5	86.5	27	10.7
LZ4 (compression level 9)	60	47	61	23.5	93	89	39	14.5
LZMA (compression level 9)	57.8	52	34	12.4	88.7	87.5	27	10.1

Table 25: The benchmark raw data results and schemas for the plot in Figure 28.

Serialization Format	Schema	Uncompressed	GZIP	LZ4	LZMA
ASN.1 (PER Unaligned)	`schema.asn`	21	41	40	45
Apache Avro (unframed)	`schema.json`	24	44	43	47
Microsoft Bond (Compact Binary v1)	`schema.bond`	30	50	49	54
Cap'n Proto (Binary Encoding)	`schema.capnp`	96	76	97	72
Cap'n Proto (Packed Encoding)	`schema.capnp`	44	64	63	68
FlatBuffers	`schema.fbs`	100	80	101	79
Protocol Buffers (Binary Wire Format)	`schema.proto`	23	43	42	47
Apache Thrift (Compact Protocol)	`schema.thrift`	26	46	45	50
JSON	-	82	88	96	90
BSON	-	102	103	117	100
CBOR	-	61	76	78	76
FlexBuffers	-	89	102	107	103
MessagePack	-	61	77	78	78
Smile	-	70	86	88	88
UBJSON	-	74	87	90	87

In Figure 29, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications.

In terms of compression, LZ4 results in the lower median for schema-driven binary serialization specifications while GZIP results in the lower median for schema-less binary serialization specifications. However, compression is not space-efficient in terms of the median for both schema-driven and schema-less binary serialization specifications. While compression does not contribute to space-efficiency, it reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers for schema-driven binary serialization specifications is LZMA, the compression formats with the smaller inter-quartile range for schema-driven binary serialization specifications are GZIP and LZMA, the compression format with the smaller range between the upper and lower whiskers for schema-less binary serialization specifi-

Figure 29: Box plot of the statistical results in Table 24.

cations is GZIP, and the compression format with the smaller inter-quartile range for schema-less binary serialization specifications is LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that compression does not contribute to space-efficiency in comparison to both uncompressed schema-driven and schema-less binary serialization specifications.

6.9 SAP Cloud SDK Continuous Delivery Toolkit Configuration

SAP Cloud SDK [72] is a framework that includes support for continuous integration and delivery pipelines to develop applications for the SAP [73] enterprise resource planning platform used by industries such as finance, healthcare and retail. In Figure 30, we demonstrate a **Tier 1 minified $<$ 100 bytes boolean redundant flat** (Tier 1 BRF from Table 2) JSON document that defines a blank pipeline with no declared steps.

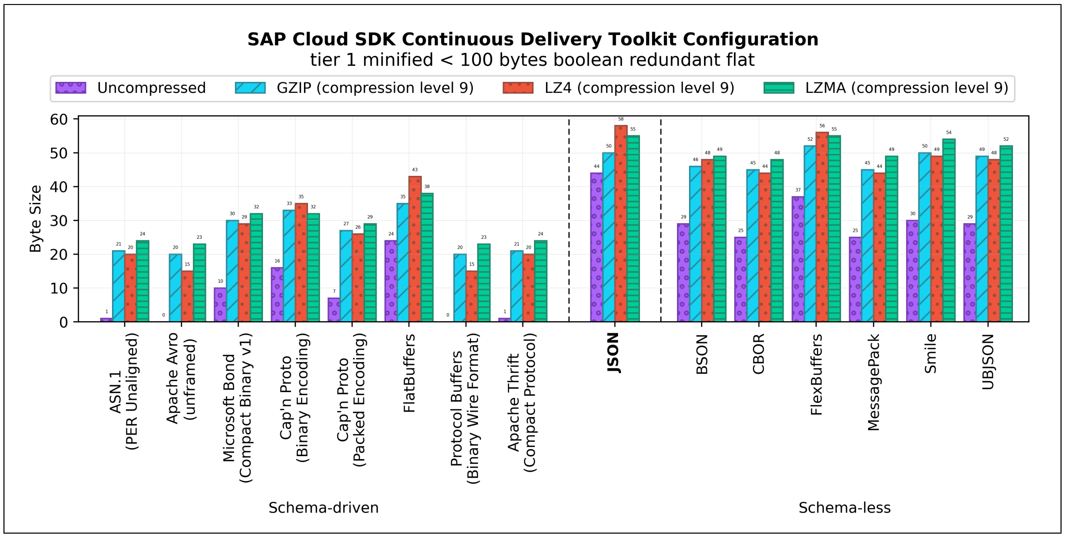

Figure 30: The benchmark results for the SAP Cloud SDK Continuous Delivery Toolkit Configuration test case listed in Table 4 and Table 5.

The smallest bit-string is produced by both Apache Avro [27] and Protocol Buffers [31] (0 bytes), closely followed by both ASN.1 PER Unaligned [57] and Apache Thrift [60] (1 byte), and Cap'n Proto Packed Encoding [69] (7 bytes). Apache Avro and Protocol Buffers achieve a zero byte-size as the input document consists of a set of *null* values, which these serialization specifications represent by not encoding the corresponding fields. The binary serialization specifications that produced the smallest bit-strings are schema-driven, and with the exception of Cap'n Proto Packed Encoding, which takes the fifth place, they are also sequential. Conversely, the largest bit-string is produced by FlexBuffers [67] (37 bytes), followed by Smile [55] (30 bytes) and both BSON [42] and UBJSON [7] (29 bytes). The binary serialization specifications that produced the largest bit-strings are schema-less and with the exception of FlexBuffers, they are also sequential [71]. In comparison to JSON [17] (44 bytes), binary serialization achieves 0 bytes in the best case for this input document, the maximal possible size reduction. Additionally, none of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON.

For this Tier 1 BRF document, the best performing schema-less serialization specifications are CBOR [5] and MessagePack [29] (25 bytes). As shown in Table 26, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. The entire selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing pointer-based serialization specification is Cap'n Proto Packed Encoding [69] (7 bytes).

The compression formats listed in subsection 5.5 do not result in positive gains for any bit-string. The best performing compression format for JSON is GZIP [16] (50 bytes).

[72] `https://sap.github.io/cloud-sdk/`

[73] `https://www.sap.com/index.html`

In Figure 31, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications.

In terms of compression, LZ4 results in the lower medians for schema-driven binary serialization specifications while GZIP results in the lower median for schema-less binary serialization specifications. However, compression is not space-efficient in terms of the median for both schema-driven and schema-less binary serialization specifications. Additionally, the use of LZ4 for schema-less binary serialization specifications exhibits upper outliers. While compression does not contribute to space-efficiency, it reduces the range between the upper and lower whiskers and inter-quartile range for schema-driven binary serialization specifications. In particular, the compression formats with the smaller range between the upper and lower whiskers for schema-driven binary serialization specifications are GZIP and LZMA, the compression format with the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA, and the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-less binary serialization specifications is LZ4.

Table 26: A byte-size statistical analysis of the benchmark results shown in Figure 30 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	7.4	4	24	8.3	29.2	29	12	4.0
GZIP (compression level 9)	25.9	24	15	5.8	47.8	47.5	7	2.7
LZ4 (compression level 9)	25.4	23	28	9.3	48.2	48	12	4.0
LZMA (compression level 9)	28.1	26.5	15	5.2	51.2	50.5	7	2.7

Table 27: The benchmark raw data results and schemas for the plot in Figure 30.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	1	21	20	24
Apache Avro (unframed)	`schema.json`	0	20	15	23
Microsoft Bond (Compact Binary v1)	`schema.bond`	10	30	29	32
Cap'n Proto (Binary Encoding)	`schema.capnp`	16	33	35	32
Cap'n Proto (Packed Encoding)	`schema.capnp`	7	27	26	29
FlatBuffers	`schema.fbs`	24	35	43	38
Protocol Buffers (Binary Wire Format)	`schema.proto`	0	20	15	23
Apache Thrift (Compact Protocol)	`schema.thrift`	1	21	20	24
JSON	-	44	50	58	55
BSON	-	29	46	48	49
CBOR	-	25	45	44	48
FlexBuffers	-	37	52	56	55
MessagePack	-	25	45	44	49
Smile	-	30	50	49	54
UBJSON	-	29	49	48	52

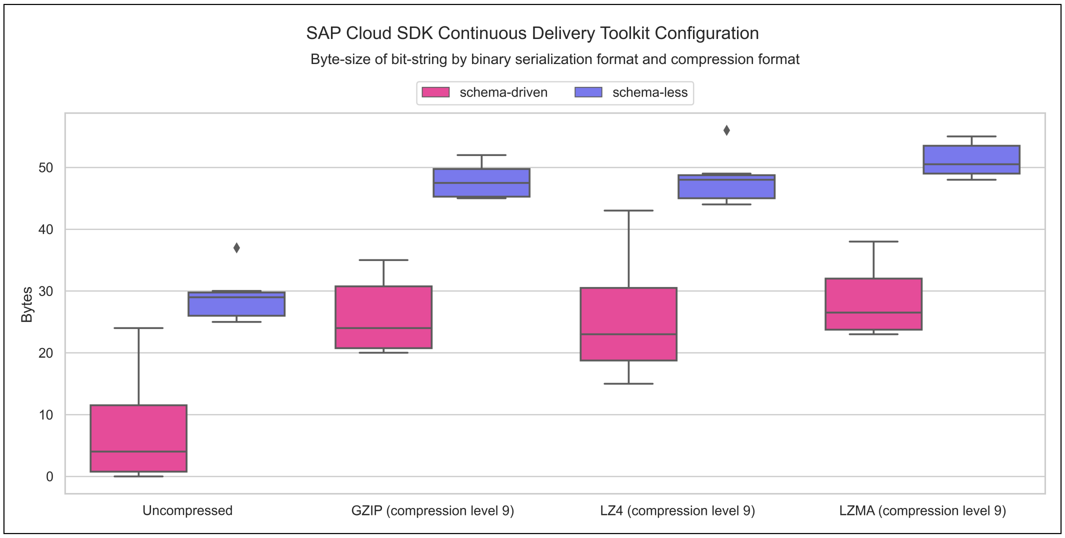

Figure 31: Box plot of the statistical results in Table 26.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that compression does not contribute to space-efficiency in comparison to both uncompressed schema-driven and schema-less binary serialization specifications.

6.10 TSLint Linter Definition (Multi-rule)

TSLint [74] is now an obsolete open-source linter for the TypeScript [75] programming language. TSLint was created by the Big Data analytics company Palantir [76] and was merged with the ESLint open-source JavaScript linter in 2019 [77]. In Figure 32, we demonstrate a **Tier 1 minified $<$ 100 bytes boolean redundant nested** (Tier 1 BRN from Table 2) JSON document that consists of a TSLint configuration that enables and configures a set of built-in rules.

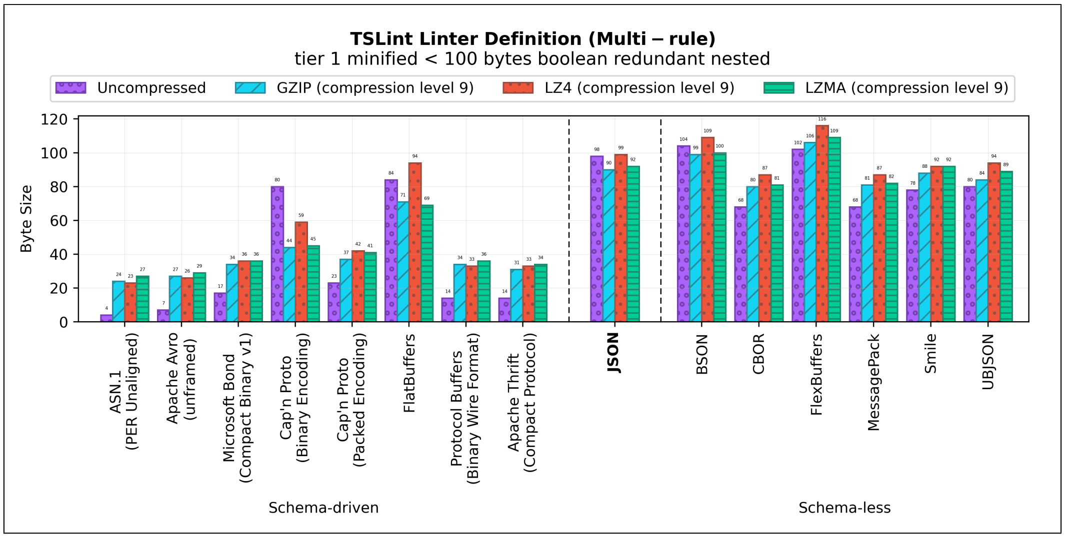

Figure 32: The benchmark results for the TSLint Linter Definition (Multi-rule) test case listed in Table 4 and Table 5.

The smallest bit-string is produced by ASN.1 PER Unaligned [57] (4 bytes), followed by Apache Avro [27] (7 bytes) and both Protocol Buffers [31] and Apache Thrift [60] (14 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by BSON [42] (104 bytes), closely followed by FlexBuffers [67] (102 bytes) and FlatBuffers [66] (84 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are pointer-based [71]. In comparison to JSON [17] (98 bytes), binary serialization achieves a **24.5x** size reduction in the best case for this input document. Similar large size reductions are observed in JSON documents whose content is dominated by *boolean* and *numeric* values. However, 2 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: BSON [42] and FlexBuffers [67]. These binary serialization specifications are schema-less.

For this Tier 1 BRN document, the best performing schema-driven serialization specification achieves a **17x** size reduction compared to the best performing schema-less serialization specification: CBOR [5] and MessagePack [29] (68 bytes). As shown in Table 28, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification achieves a **5.7x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (23 bytes).

[74] `https://palantir.github.io/tslint`

[75] `https://www.typescriptlang.org`

[76] `https://www.palantir.com`

[77] `https://github.com/palantir/tslint/issues/4534`

The compression formats listed in subsection 5.5 result in positive gains for the bit-strings produced by Cap'n Proto Binary Encoding [69], FlatBuffers [66], JSON [17] and BSON [42]. The best performing uncompressed binary serialization specification achieves a **22.5x** size reduction compared to the best performing compression format for JSON: GZIP [16] (90 bytes).

Table 28: A byte-size statistical analysis of the benchmark results shown in Figure 32 divided by schema-driven and schema-less specifications.

Category	Schema-driven				Schema-less			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	30.4	15.5	80	30.3	83.3	79	36	14.6
GZIP (compression level 9)	37.8	34	47	13.8	89.7	86	26	9.6
LZ4 (compression level 9)	43.3	34.5	71	21.8	97.5	93	29	11.1
LZMA (compression level 9)	39.6	36	42	12.4	92.2	90.5	28	9.9

Table 29: The benchmark raw data results and schemas for the plot in Figure 32.

Serialization Format	Schema	Uncompressed	GZIP	LZ4	LZMA
ASN.1 (PER Unaligned)	`schema.asn`	4	24	23	27
Apache Avro (unframed)	`schema.json`	7	27	26	29
Microsoft Bond (Compact Binary v1)	`schema.bond`	17	34	36	36
Cap'n Proto (Binary Encoding)	`schema.capnp`	80	44	59	45
Cap'n Proto (Packed Encoding)	`schema.capnp`	23	37	42	41
FlatBuffers	`schema.fbs`	84	71	94	69
Protocol Buffers (Binary Wire Format)	`schema.proto`	14	34	33	36
Apache Thrift (Compact Protocol)	`schema.thrift`	14	31	33	34
JSON	-	98	90	99	92
BSON	-	104	99	109	100
CBOR	-	68	80	87	81
FlexBuffers	-	102	106	116	109
MessagePack	-	68	81	87	82
Smile	-	78	88	92	92
UBJSON	-	80	84	94	89

In Figure 33, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers of uncompressed schema-driven binary serialization specifications is smaller than the range between the upper and lower whiskers of uncompressed schema-less binary serialization specifications. However, the inter-quartile range of both both uncompressed schema-driven and schema-less binary serialization specifications is similar.

In terms of compression, GZIP results in the lower median for both schema-driven and schema-less binary serialization specifications. However, compression is not space-efficient in terms of the median for both schema-driven and schema-less binary serialization specifications. Additionally, the use of GZIP, LZ4 and LZMA for schema-driven binary serialization specifications exhibits upper outliers. While compression does not contribute to space-efficiency, it reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers for schema-driven binary serialization specifications is LZMA, the compression

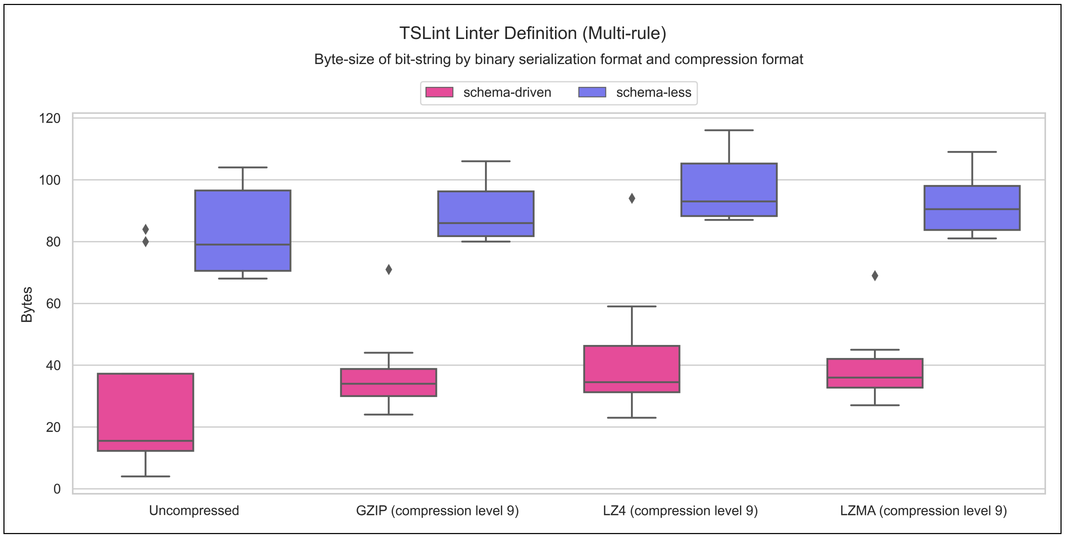

Figure 33: Box plot of the statistical results in Table 28.

formats with the smaller inter-quartile range for schema-driven binary serialization specifications are GZIP and LZMA, the compression format with the smaller range between the upper and lower whiskers for schema-less binary serialization specifications is GZIP, and the compression formats with the smaller inter-quartile range for schema-less binary serialization specifications are GZIP and LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that compression does not contribute to space-efficiency in comparison to both uncompressed schema-driven and schema-less binary serialization specifications.

6.11 CommitLint Configuration (Basic)

CommitLint [78] is an open-source command-line tool to enforce version-control commit conventions in software engineering projects. CommitLint is a community effort under the Conventional Changelog [79] organization formed by employees from companies including GitHub [80] and Google [81]. In Figure 34, we demonstrate a **Tier 1 minified $<$ 100 bytes boolean non-redundant flat** (Tier 1 BNF from Table 2) JSON document that represents a CommitLint configuration file which declares that CommitLint must not use its default commit ignore rules.

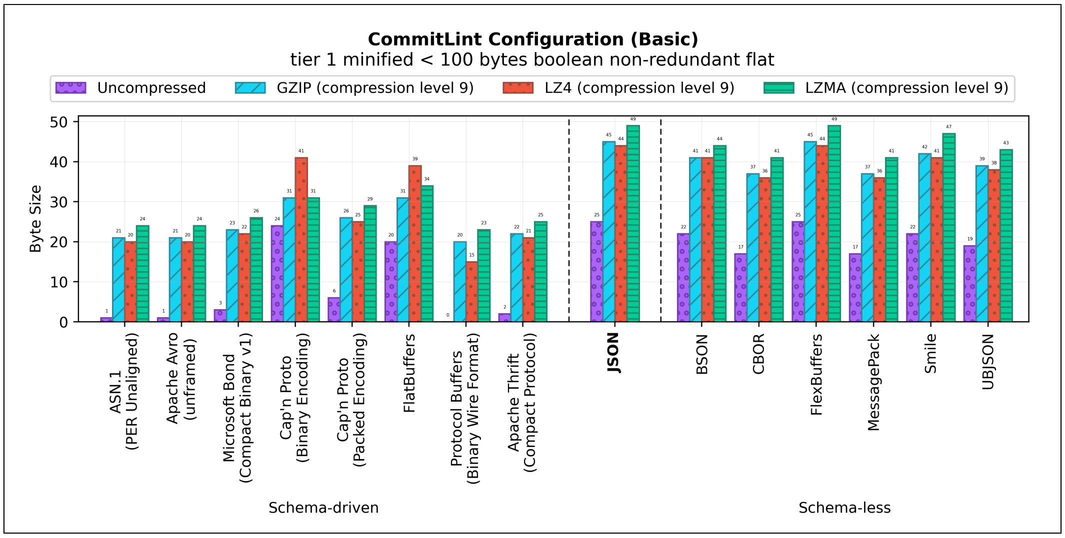

Figure 34: The benchmark results for the CommitLint Configuration (Basic) test case listed in Table 4 and Table 5.

The smallest bit-string is produced by Protocol Buffers [31] (0 bytes), closely followed by both ASN.1 PER Unaligned [57] and Apache Avro [27] (1 byte), and Apache Thrift [60] (2 bytes). Protocol Buffers [31] achieves a zero byte-size as the input document consists of the boolean value *false*, which Protocol Buffers represents by not encoding the corresponding field. The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the larges bit-string is produced by FlexBuffers [67] (152 bytes) followed by UBJSON [7] (137 bytes) and BSON [42] (133 bytes). The binary serialization specifications that produced the largest bit-strings are schema-less. With the exception of FlexBuffers, the binary serialization specifications that produced the largest bit-strings are also sequential [71]. In comparison to JSON [17] (25 bytes), binary serialization achieves 0 bytes in the best case for this input document, the maximal possible size reduction. Additionally, none of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON. However, the FlexBuffers [67] schema-less pointer-based serialization specification result in a bit-string that is equal in size to JSON.

For this Tier 1 BNF document, the best performing schema-less serialization specifications are CBOR [5] and MessagePack [29] (17 bytes). As shown in Table 30, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less

[78] `https://commitlint.js.org/#/`
[79] `https://github.com/conventional-changelog`
[80] `https://github.com/zeke`
[81] `https://github.com/bcoe`

counterparts listed in Table 7. The best performing pointer-based serialization specification is Cap'n Proto Packed Encoding [69] (6 bytes).

The compression formats listed in subsection 5.5 do not result in positive gains for any bit-string due to the overhead of encoding the dictionary data structures and the low redundancy of the input data. The best performing compression format for JSON is LZ4 (44 bytes).

Table 30: A byte-size statistical analysis of the benchmark results shown in Figure 34 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	7.1	2.5	24	8.8	20.3	20.5	8	2.9
GZIP (compression level 9)	24.4	22.5	11	4.2	40.2	40	8	2.9
LZ4 (compression level 9)	25.4	21.5	26	8.8	39.3	39.5	8	2.9
LZMA (compression level 9)	27	25.5	11	3.7	44.2	43.5	8	3.0

Table 31: The benchmark raw data results and schemas for the plot in Figure 34.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	1	21	20	24
Apache Avro (unframed)	`schema.json`	1	21	20	24
Microsoft Bond (Compact Binary v1)	`schema.bond`	3	23	22	26
Cap'n Proto (Binary Encoding)	`schema.capnp`	24	31	41	31
Cap'n Proto (Packed Encoding)	`schema.capnp`	6	26	25	29
FlatBuffers	`schema.fbs`	20	31	39	34
Protocol Buffers (Binary Wire Format)	`schema.proto`	0	20	15	23
Apache Thrift (Compact Protocol)	`schema.thrift`	2	22	21	25
JSON	-	25	45	44	49
BSON	-	22	41	41	44
CBOR	-	17	37	36	41
FlexBuffers	-	25	45	44	49
MessagePack	-	17	37	36	41
Smile	-	22	42	41	47
UBJSON	-	19	39	38	43

In Figure 35, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications.

In terms of compression, LZ4 results in the lower median for both schema-driven and schema-less binary serialization specifications. However, compression is not space-efficient in terms of the median for both schema-driven and schema-less binary serialization specifications. While compression does not contribute to space-efficiency, it reduces the range between the upper and lower whiskers and inter-quartile range for schema-driven binary serialization specifications. In particular, the compression formats with the smaller range between the upper and lower whiskers for schema-driven binary

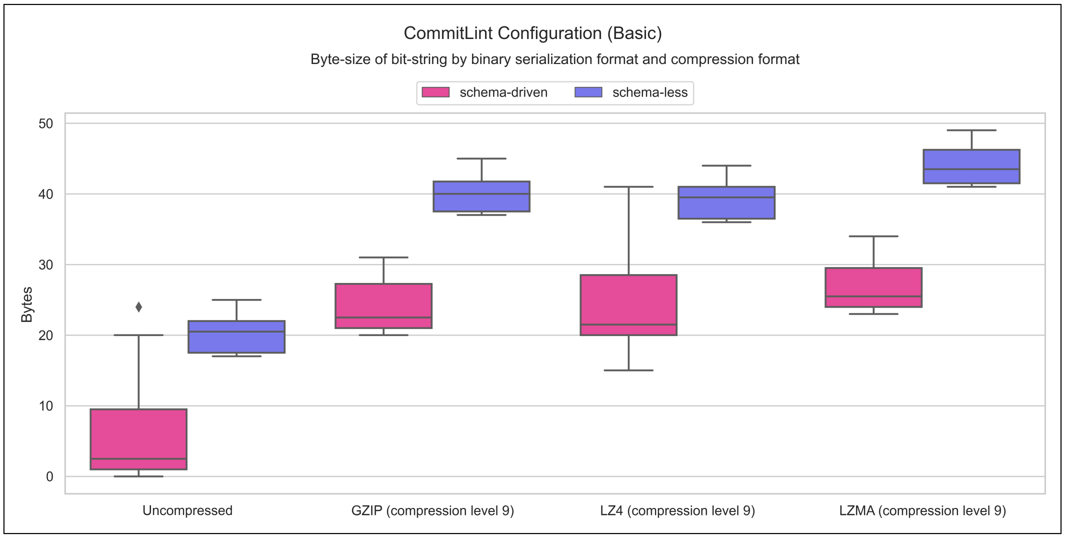

Figure 35: Box plot of the statistical results in Table 30.

serialization specifications are GZIP and LZMA, the compression format with the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA, and the compression format with the smaller inter-quartile range for schema-less binary serialization specifications is GZIP.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that compression does not contribute to space-efficiency in comparison to both uncompressed schema-driven and schema-less binary serialization specifications.

6.12 TSLint Linter Definition (Basic)

TSLint [82] is now an obsolete open-source linter for the TypeScript [83] programming language. TSLint was created by the Big Data analytics company Palantir [84] and was merged with the ESLint open-source JavaScript linter in 2019 [85]. In Figure 36, we demonstrate a **Tier 1 minified $<$ 100 bytes boolean non-redundant nested** (Tier 1 BNN from Table 2) JSON document that consists of a basic TSLint configuration that enforces grouped alphabetized imports.

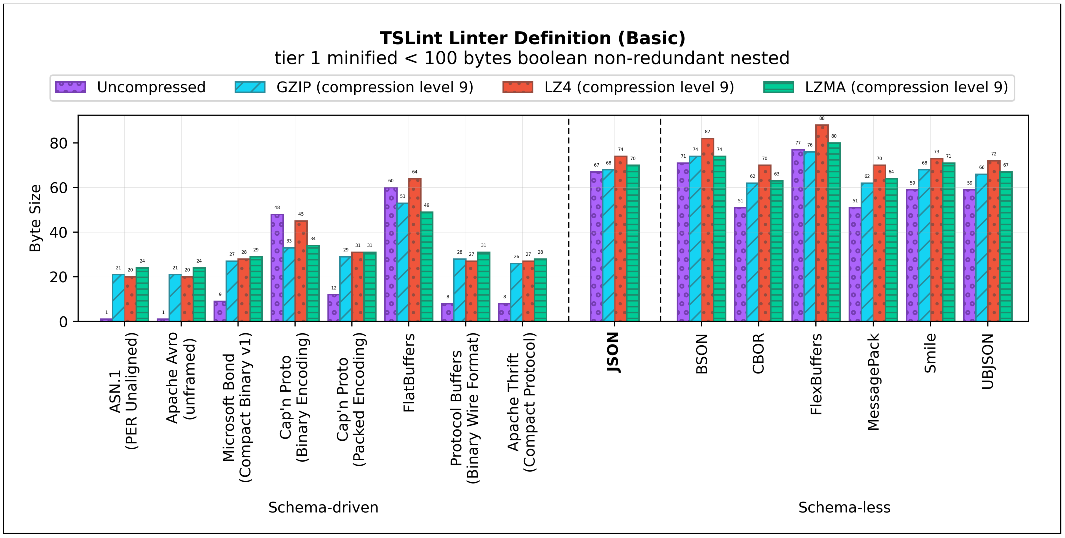

Figure 36: The benchmark results for the TSLint Linter Definition (Basic) test case listed in Table 4 and Table 5.

The smallest bit-string is produced by both ASN.1 PER Unaligned [57] and Apache Avro [27] (1 byte), followed by both Protocol Buffers [31] and Apache Thrift [60] (8 bytes), and Microsoft Bond [43] (9 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by FlexBuffers [67] (77 bytes), closely followed by BSON [42] (71 bytes) and FlatBuffers [66] (60 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are pointer-based [71]. In comparison to JSON [17] (67 bytes), binary serialization achieves a **67x** size reduction in the best case for this input document. Similar large size reductions are observed in JSON documents whose content is dominated by *boolean* and *numeric* values. However, 2 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: BSON [42] and FlexBuffers [67]. These binary serialization specifications are schema-less.

For this Tier 1 BNN document, the best performing schema-driven serialization specification achieves a **51x** size reduction compared to the best performing schema-less serialization specification: MessagePack [29] (51 bytes). As shown in Table 32, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specification FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification achieves a **12x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (12 bytes).

[82] `https://palantir.github.io/tslint`
[83] `https://www.typescriptlang.org`
[84] `https://www.palantir.com`
[85] `https://github.com/palantir/tslint/issues/4534`

The compression formats listed in subsection 5.5 result in positive gains for the bit-strings produced by Cap'n Proto Binary Encoding [69], FlatBuffers [66] and FlexBuffers [67]. The best performing uncompressed binary serialization specification achieves a **68x** size reduction compared to the best performing compression format for JSON: GZIP [16] (68 bytes).

Table 32: A byte-size statistical analysis of the benchmark results shown in Figure 36 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	18.4	8.5	59	21.1	61.3	59	26	9.7
GZIP (compression level 9)	29.8	27.5	32	9.5	68	67	14	5.4
LZ4 (compression level 9)	32.8	27.5	44	13.9	75.8	72.5	18	6.8
LZMA (compression level 9)	31.3	30	25	7.4	69.8	69	17	5.9

Table 33: The benchmark raw data results and schemas for the plot in Figure 36.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	1	21	20	24
Apache Avro (unframed)	`schema.json`	1	21	20	24
Microsoft Bond (Compact Binary v1)	`schema.bond`	9	27	28	29
Cap'n Proto (Binary Encoding)	`schema.capnp`	48	33	45	34
Cap'n Proto (Packed Encoding)	`schema.capnp`	12	29	31	31
FlatBuffers	`schema.fbs`	60	53	64	49
Protocol Buffers (Binary Wire Format)	`schema.proto`	8	28	27	31
Apache Thrift (Compact Protocol)	`schema.thrift`	8	26	27	28
JSON	-	67	68	74	70
BSON	-	71	74	82	74
CBOR	-	51	62	70	63
FlexBuffers	-	77	76	88	80
MessagePack	-	51	62	70	64
Smile	-	59	68	73	71
UBJSON	-	59	66	72	67

In Figure 37, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers of uncompressed schema-driven binary serialization specifications is smaller than the range between the upper and lower whiskers of uncompressed schema-less binary serialization specifications. However, the inter-quartile range of both both uncompressed schema-driven and schema-less binary serialization specifications is similar.

In terms of compression, GZIP and LZ4 result in the lower medians for schema-driven binary serialization specifications while GZIP results in the lower median for schema-less binary serialization specifications. However, compression is not space-efficient in terms of the median for both schema-driven and schema-less binary serialization specifications. Additionally, the use of GZIP, LZ4 and LZMA for schema-driven binary serialization specifications exhibits upper outliers. While compression does not contribute to space-efficiency, it reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and

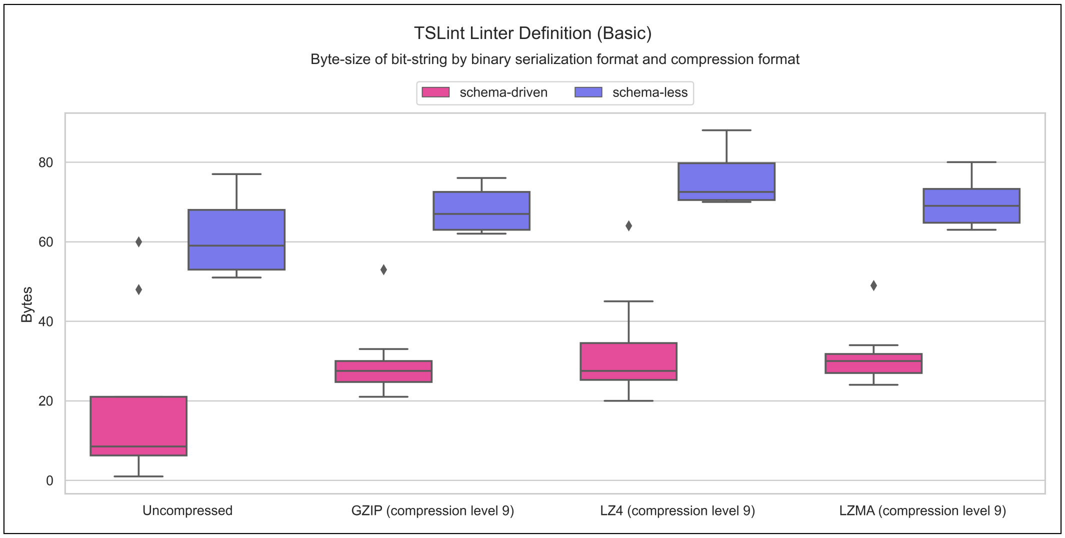

Figure 37: Box plot of the statistical results in Table 32.

lower whiskers for schema-driven binary serialization specifications is LZMA, the compression formats with the smaller inter-quartile range for schema-driven binary serialization specifications are GZIP and LZMA, the compression format with the smaller range between the upper and lower whiskers for schema-less binary serialization specifications is GZIP, and the compression formats with the smaller inter-quartile range for schema-less binary serialization specifications are LZ4 and LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that compression does not contribute to space-efficiency in comparison to both uncompressed schema-driven and schema-less binary serialization specifications.

6.13 GeoJSON Example Document

GeoJSON [11] is a standard to encode geospatial information using JSON. GeoJSON is used in industries that have geographical and geospatial use cases such as engineering, logistics and telecommunications. In Figure 38, we demonstrate a **Tier 2 minified $\geq$ 100 $<$ 1000 bytes numeric redundant nested** (Tier 2 NRN from Table 2) JSON document that defines an example polygon using the GeoJSON format.

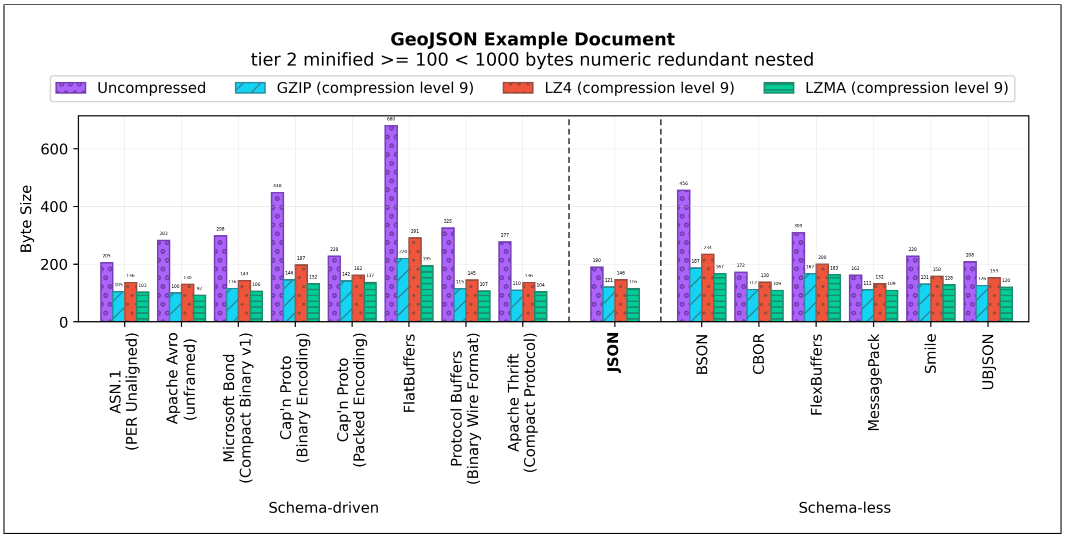

Figure 38: The benchmark results for the GeoJSON Example Document test case listed in Table 4 and Table 5.

In this exceptional case, the smallest bit-strings for this input document are produced by the schema-less sequential specifications MessagePack [29] (162 bytes) and CBOR [5] (172 bytes), followed by the schema-driven sequential specification ASN.1 PER Unaligned [57] (205 bytes). In comparison to other input documents, this input document defines poly-dimensional JSON [17] arrays, which the schema-driven binary serialization specifications from the selection do not encode in a space-efficient manner. Similarly to other cases, the largest bit-string is produced by FlatBuffers [66] (680 bytes), followed by BSON [42] (456 bytes) and Cap'n Proto Binary Encoding [69] (448 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are pointer-based [71]. In comparison to JSON [17] (190 bytes), binary serialization only achieves a **1.1x** size reduction in the best case for this input document. With the exception of the schema-less sequential MessagePack [29] and CBOR [5] binary serialization specifications, all the other JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON.

For this Tier 2 NRN document, the smaller bit-string is produced by a schema-less specification. However, the best performing schema-less serialization specification only achieves a **1.2x** size reduction compared to the best performing schema-driven serialization specification: ASN.1 PER Unaligned [57] (205 bytes). As shown in Table 34, uncompressed schema-less specifications provide smaller *average* and *median* bit-strings than uncompressed schema-driven specifications. Additionally, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. The schema-less and sequential specifications CBOR [5] and MessagePack [29] produce bit-strings that are smaller to all their schema-driven counterparts listed in Table 6. The best performing sequential serialization specification only achieves a **1.4x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (228 bytes).

The compression formats listed in subsection 5.5 result in positive gains for all bit-strings. The best performing compression format for JSON, LZMA (116 bytes), achieve a **1.3x** size reduction compared to the best performing uncompressed binary serialization specification.

In Figure 39, contrary to other cases, we observe the medians for uncompressed schema-less binary serialization specifications to be smaller in comparison to uncompressed schema-driven binary serialization specifications. The range between the upper and lower whiskers of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers of uncompressed schema-driven binary serialization specifications. However, the inter-quartile range of uncompressed schema-less binary serialization specifications is larger than the inter-quartile range of uncompressed schema-driven binary serialization specifications. Additionally, their respective quartiles overlap.

In terms of compression, LZMA results in the lower median for both schema-driven and schema-less binary serialization specifications. Additionally, GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to both uncompressed schema-driven and schema-less binary serialization specifications. However, the use of GZIP, LZ4 and LZMA for schema-driven binary serialization specifications exhibits upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression formats with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-driven binary serialization specifications are GZIP and LZMA, the compression format with the smaller range between the upper and lower whiskers for schema-less binary serialization specifications is LZMA, and the compression

Table 34: A byte-size statistical analysis of the benchmark results shown in Figure 38 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	343	290.5	475	144.6	255.8	218	294	101.5
GZIP (compression level 9)	131.8	115.5	120	36.8	139	128.5	76	28.4
LZ4 (compression level 9)	167.5	144	161	50.8	169.2	155.5	102	36.3
LZMA (compression level 9)	122	106.5	103	31.1	132.7	124	58	23.8

Table 35: The benchmark raw data results and schemas for the plot in Figure 38.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	205	105	136	103
Apache Avro (unframed)	`schema.json`	283	100	130	92
Microsoft Bond (Compact Binary v1)	`schema.bond`	298	116	143	106
Cap'n Proto (Binary Encoding)	`schema.capnp`	448	146	197	132
Cap'n Proto (Packed Encoding)	`schema.capnp`	228	142	162	137
FlatBuffers	`schema.fbs`	680	220	291	195
Protocol Buffers (Binary Wire Format)	`schema.proto`	325	115	145	107
Apache Thrift (Compact Protocol)	`schema.thrift`	277	110	136	104
JSON	-	190	121	146	116
BSON	-	456	187	234	167
CBOR	-	172	112	138	109
FlexBuffers	-	309	167	200	163
MessagePack	-	162	111	132	109
Smile	-	228	131	158	128
UBJSON	-	208	126	153	120

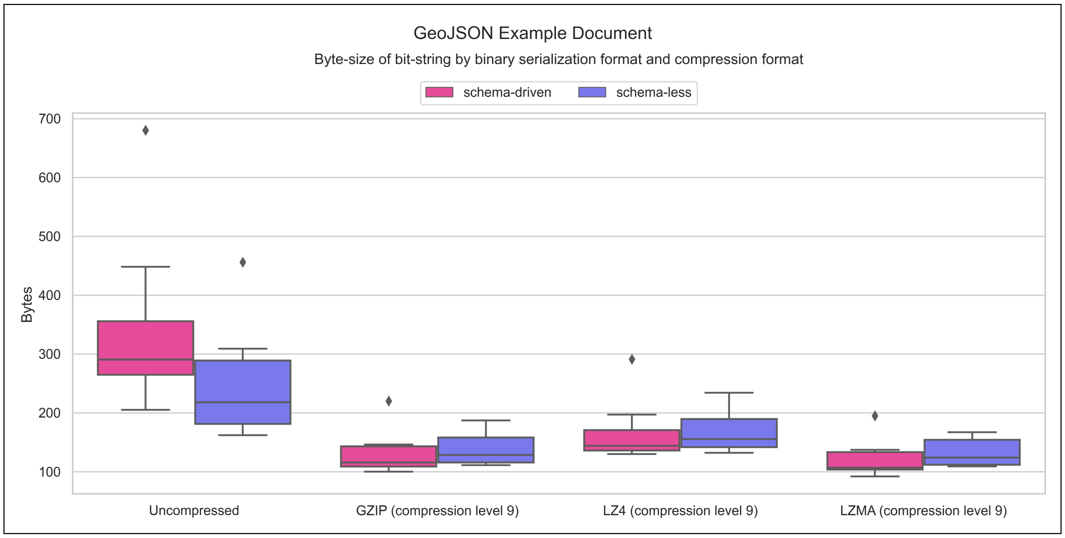

Figure 39: Box plot of the statistical results in Table 34.

formats with the smaller inter-quartile range for schema-less binary serialization specifications are GZIP and LZMA.

Overall, we conclude that uncompressed schema-less binary serialization specifications are space-efficient in comparison to uncompressed schema-driven binary serialization specifications and that all the considered compression formats are space-efficient in comparison to uncompressed schema-driven and schema-less binary serialization specifications.

6.14 OpenWeatherMap API Example Document

OpenWeatherMap [86] is a weather data and forecast API provider used in industries such as energy, agriculture, transportation and construction. In Figure 40, we demonstrate a **Tier 2 minified** $\geq$ **100** $<$ **1000 bytes numeric non-redundant flat** (Tier 2 NNF from Table 2) JSON document that consists of an HTTP/1.1 [25] response of the weather information in Mountain View, California on June 12, 2019 at 2:44:05 PM GMT.

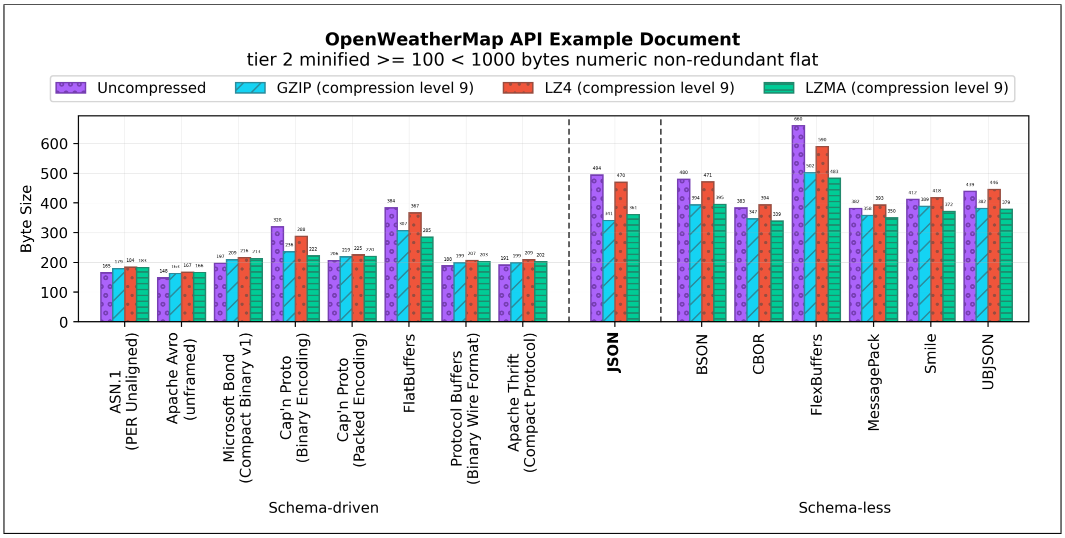

Figure 40: The benchmark results for the OpenWeatherMap API Example Document test case listed in Table 4 and Table 5.

The smallest bit-string is produced by Apache Avro [27] (148 bytes), followed by ASN.1 PER Unaligned [57] (165 bytes) and Protocol Buffers [31] (188 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by FlexBuffers [67] (660 bytes), followed by BSON [42] (480 bytes) and UBJSON [7] (439 bytes). The binary serialization specifications that produced the largest bit-strings are schema-less and with the exception of FlexBuffers, they are also sequential [71]. In comparison to JSON [17] (494 bytes), binary serialization achieves a **3.3x** size reduction in the best case for this input document. However, 1 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: the schema-less and pointer-based FlexBuffers [67] binary serialization specification.

For this Tier 2 NNF document, the best performing schema-driven serialization specification achieves a **2.5x** size reduction compared to the best performing schema-less serialization specification: MessagePack [29] (382 bytes). As shown in Table 36, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. Additionally, as highlighted by the *range* and *standard deviation*, uncompressed schema-less specifications exhibit higher size reduction variability given that FlexBuffers [67] produces a notably large bit-string. With the exception of the pointer-based binary serialization specification FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification only achieves a **1.3x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (206 bytes).

The compression formats listed in subsection 5.5 result in positive gains for all bit-strings except the ones produced by ASN.1 PER Unaligned [57], Apache Avro [27], Microsoft Bond [43], Cap'n Proto Packed Encoding [69], Protocol Buffers [31] and Apache Thrift [60]. The best performing uncompressed binary serialization specification achieves a **2.3x** size reduction compared to the best performing compression format for JSON: GZIP [16] (341 bytes).

[86] `https://openweathermap.org`

In Figure 41, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers of uncompressed schema-driven and schema-less binary serialization specifications is similar. However, the inter-quartile range of uncompressed schema-driven binary serialization specifications is smaller than the inter-quartile range of uncompressed schema-less binary serialization specifications.

In terms of compression, GZIP results in the lower medians for schema-driven binary serialization specifications while LZMA results in the lower median for schema-less binary serialization specifications. Compression is not space-efficient in terms of the median in comparison to uncompressed schema-driven binary serialization specifications. However, GZIP and LZMA are space-efficient in terms of the median in comparison to uncompressed schema-less binary serialization specifications. Additionally, the use of GZIP, LZ4 and LZMA for both schema-driven binary serialization specifications and schema-less binary serialization specifications exhibits upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA, and the compression format

Table 36: A byte-size statistical analysis of the benchmark results shown in Figure 40 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	224.9	194	236	77.1	459.3	425.5	278	95.9
GZIP (compression level 9)	213.9	204	144	41.0	395.3	385.5	155	50.5
LZ4 (compression level 9)	232.9	212.5	200	60.6	452	432	197	67.6
LZMA (compression level 9)	211.8	208	119	32.8	386.3	375.5	144	47.0

Table 37: The benchmark raw data results and schemas for the plot in Figure 40.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	165	179	184	183
Apache Avro (unframed)	`schema.json`	148	163	167	166
Microsoft Bond (Compact Binary v1)	`schema.bond`	197	209	216	213
Cap'n Proto (Binary Encoding)	`schema.capnp`	320	236	288	222
Cap'n Proto (Packed Encoding)	`schema.capnp`	206	219	225	220
FlatBuffers	`schema.fbs`	384	307	367	285
Protocol Buffers (Binary Wire Format)	`schema.proto`	188	199	207	203
Apache Thrift (Compact Protocol)	`schema.thrift`	191	199	209	202
JSON	-	494	341	470	361
BSON	-	480	394	471	395
CBOR	-	383	347	394	339
FlexBuffers	-	660	502	590	483
MessagePack	-	382	358	393	350
Smile	-	412	389	418	372
UBJSON	-	439	382	446	379

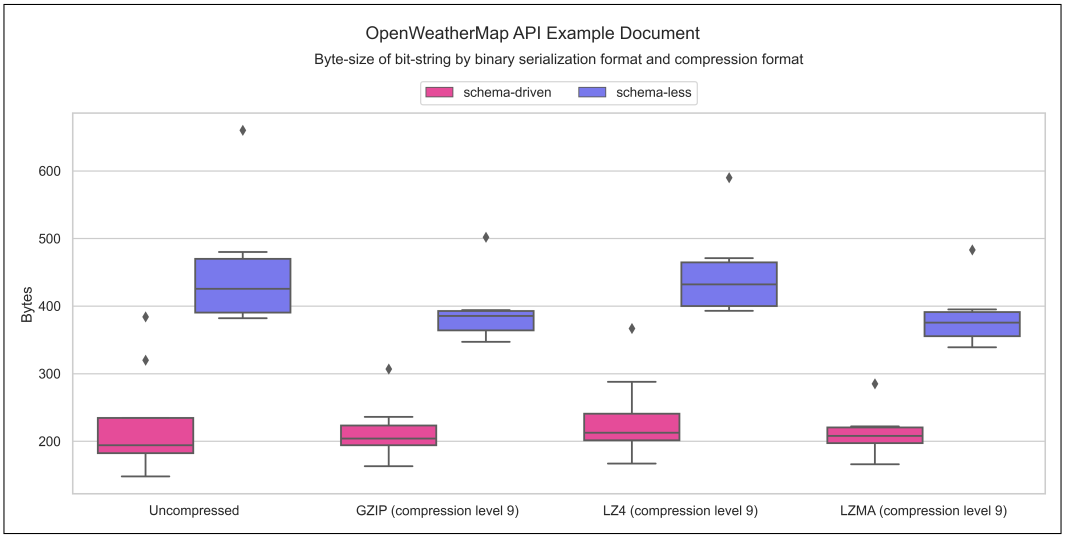

Figure 41: Box plot of the statistical results in Table 36.

with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-less binary serialization specifications is GZIP.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications. Compression does not contribute to space-efficiency in comparison to schema-driven binary serialization specifications but GZIP and LZMA are space-efficient in comparison to uncompressed schema-less binary serialization specifications.

6.15 OpenWeather Road Risk API Example

OpenWeatherMap [87] is a weather data and forecast API provider used in industries such as energy, agriculture, transportation and construction. In Figure 42, we demonstrate a **Tier 2 minified $\geq$ 100 $<$ 1000 bytes numeric non-redundant nested** (Tier 2 NNN from Table 2) JSON document that consists of an example HTTP/1.1 [25] Road Risk API response from the official API documentation that provides weather data and national alerts along a specific route.

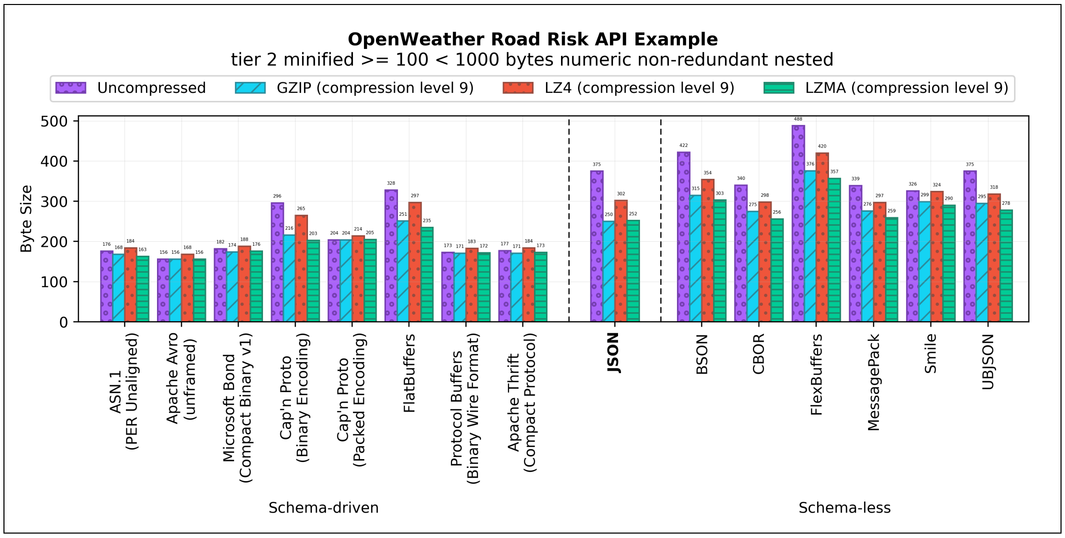

Figure 42: The benchmark results for the OpenWeather Road Risk API Example test case listed in Table 4 and Table 5.

The smallest bit-string is produced by Apache Avro [27] (156 bytes), followed by Protocol Buffers [31] (173 bytes) and ASN.1 PER Unaligned [57] (176 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by FlexBuffers [67] (488 bytes), followed by BSON [42] (422 bytes) and UBJSON [7] (375 bytes). The binary serialization specifications that produced the largest bit-strings are schema-less and with the exception of FlexBuffers, they are also sequential [71]. In comparison to JSON [17] (375 bytes), binary serialization achieves a **2.4x** size reduction in the best case for this input document. However, 2 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: BSON [42] and FlexBuffers [67]. These binary serialization specifications are schema-less. Additionally, the schema-less and sequential UBJSON [7] serialization specification results in a bit-string that is equal to JSON in size.

For this Tier 2 NNN document, the best performing schema-driven serialization specification achieves a **2x** size reduction compared to the best performing schema-less serialization specification: Smile [55] (326 bytes). As shown in Table 38, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specification FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification only achieves a **1.3x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (204 bytes).

The compression formats listed in subsection 5.5 result in positive gains for all bit-strings. The best performing uncompressed binary serialization specification achieves a **1.6x** size reduction compared to the best performing compression format for JSON: GZIP [16] (250 bytes).

[87] `https://openweathermap.org`

In Figure 43, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications.

In terms of compression, GZIP results in the lower medians for schema-driven binary serialization specifications while LZMA results in the lower median for schema-less binary serialization specifications. Additionally, GZIP and LZMA are space-efficient in terms of the median in comparison to uncompressed schema-driven binary serialization specifications and GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to uncompressed schema-less binary serialization specifications. However, the use of LZ4 for schema-driven binary serialization specifications and the use of GZIP, LZ4 and LZMA for schema-less binary serialization specifications exhibits upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers for schema-driven binary serialization specifications is LZMA, the compression formats with the smaller inter-quartile range for schema-driven binary serialization specifications are GZIP and LZMA, and

Table 38: A byte-size statistical analysis of the benchmark results shown in Figure 42 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	211.5	179.5	172	59.8	381.7	357.5	162	57.2
GZIP (compression level 9)	188.9	172.5	95	30.0	306	297	101	34.2
LZ4 (compression level 9)	210.4	186	129	43.2	335.2	321	123	42.4
LZMA (compression level 9)	185.4	174.5	79	24.9	290.5	284	101	34.0

Table 39: The benchmark raw data results and schemas for the plot in Figure 42.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	176	168	184	163
Apache Avro (unframed)	`schema.json`	156	156	168	156
Microsoft Bond (Compact Binary v1)	`schema.bond`	182	174	188	176
Cap'n Proto (Binary Encoding)	`schema.capnp`	296	216	265	203
Cap'n Proto (Packed Encoding)	`schema.capnp`	204	204	214	205
FlatBuffers	`schema.fbs`	328	251	297	235
Protocol Buffers (Binary Wire Format)	`schema.proto`	173	171	183	172
Apache Thrift (Compact Protocol)	`schema.thrift`	177	171	184	173
JSON	-	375	250	302	252
BSON	-	422	315	354	303
CBOR	-	340	275	298	256
FlexBuffers	-	488	376	420	357
MessagePack	-	339	276	297	259
Smile	-	326	299	324	290
UBJSON	-	375	295	318	278

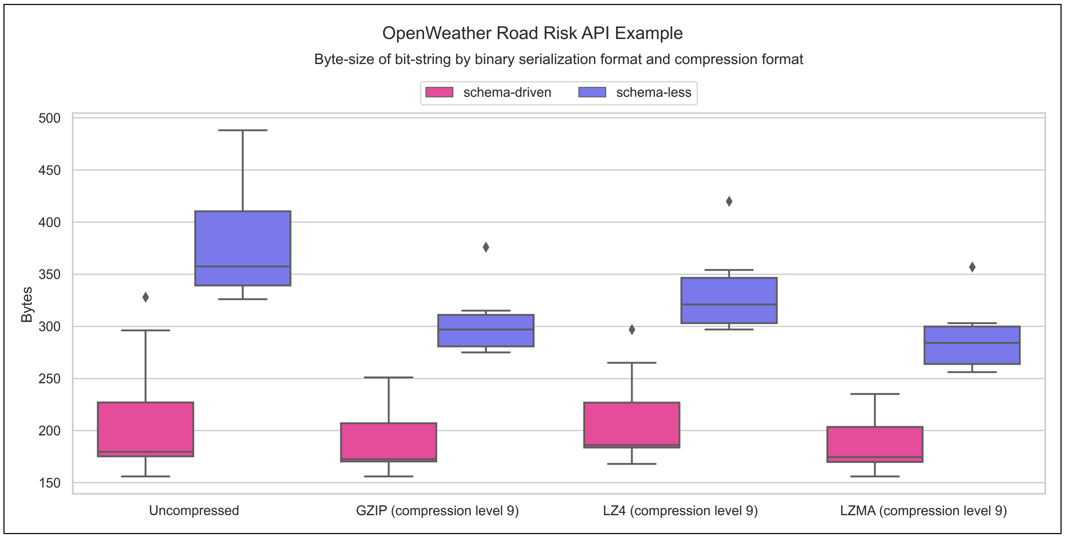

Figure 43: Box plot of the statistical results in Table 38.

the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-less binary serialization specifications is GZIP.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications. GZIP and LZMA are space-efficient in comparison to uncompressed schema-driven binary serialization specifications and all the considered compression formats are space-efficient in comparison to uncompressed schema-less binary serialization specifications.

6.16 TravisCI Notifications Configuration

TravisCI [88] is a commercial cloud-provider of continuous integration and deployment pipelines used by a wide range of companies in the software development industry such as ZenDesk, BitTorrent, and Engine Yard. In Figure 44, we demonstrate a **Tier 2 minified $\geq$ 100 $<$ 1000 bytes textual redundant flat** (Tier 2 TRF from Table 2) JSON document that consists of an example pipeline configuration for TravisCI that declares a set of credentials to post build notifications to various external services.

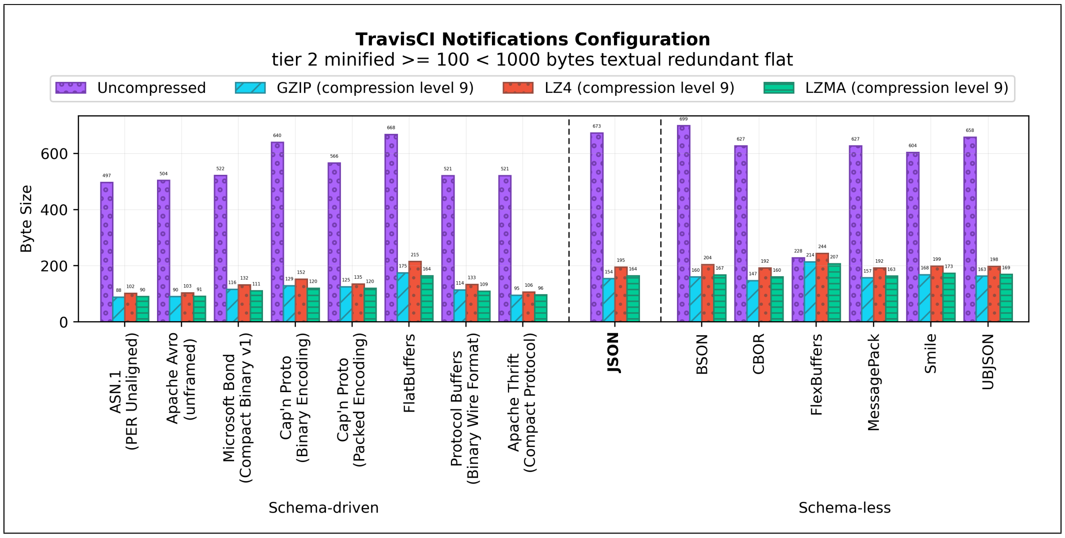

Figure 44: The benchmark results for the TravisCI Notifications Configuration test case listed in Table 4 and Table 5.

In this exceptional case, the smallest bit-string for this input document is produced by the schema-less pointer-based serialization specification FlexBuffers [67] (228 bytes), followed by the schema-driven sequential specifications ASN.1 PER Unaligned [57] (497 bytes) and Apache Avro [27] (504 bytes). In comparison to other specifications, FlexBuffers detects and de-duplicates the 7 instances of the same string value from this input document. Conversely, the largest bit-string is produced by BSON [42] (699 bytes), followed by FlatBuffers [66] (668 bytes) and UBJSON [7] (658 bytes). With the exception of FlatBuffers, the binary serialization specifications that produced the largest bit-strings are schema-less and sequential [71]. In comparison to JSON [17] (673 bytes), binary serialization achieves a **2.9x** size reduction in the best case for this input document. However, 1 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: the schema-less and sequential BSON [42] binary serialization specification.

For this Tier 2 TRF document, the smaller bit-string is produced by a schema-less specification. The best performing schema-less serialization specification achieves a **2.1x** size reduction compared to the best performing schema-driven serialization specification: ASN.1 PER Unaligned [57] (497 bytes). As shown in Table 40, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. Additionally, as highlighted by the *range* and *standard deviation*, uncompressed schema-less specifications exhibit higher size reduction variability given that FlexBuffers [67] produces a notably small bit-string. The schema-less and pointer-based specification FlexBuffers [67] produces bit-strings that are smaller to all their schema-driven counterparts listed in Table 6. The best performing pointer-based serialization specification achieves a **2.1x** size reduction compared to the best performing sequential serialization specification: ASN.1 PER Unaligned [57] (497 bytes).

The compression formats listed in subsection 5.5 result in positive gains for all bit-strings. The best performing compression format for JSON, GZIP [16] (154 bytes), achieves a **1.4x** size reduction compared to the best performing uncompressed binary serialization specification.

[88] `https://travis-ci.com`

In Figure 45, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications. The low outlier for uncompressed schema-less binary serialization specification represents the space-efficiency of FlexBuffers [67] given its string de-duplication features.

In terms of compression, LZMA results in the lower medians for schema-driven binary serialization specifications while GZIP results in the lower median for schema-less binary serialization specifications. Additionally, GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to both uncompressed schema-driven and schema-less binary serialization specifications. However, the use of GZIP, LZ4 and LZMA for schema-driven binary serialization specifications and schema-less binary serialization specifications exhibits upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA, and the compression formats with the smaller range

Table 40: A byte-size statistical analysis of the benchmark results shown in Figure 44 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	554.9	521.5	171	60.7	573.8	627	471	157.5
GZIP (compression level 9)	116.5	115	87	26.5	168.2	161.5	67	21.5
LZ4 (compression level 9)	134.8	132.5	113	34.7	204.8	198.5	52	18.0
LZMA (compression level 9)	112.6	110	74	22.4	173.2	168	47	15.7

Table 41: The benchmark raw data results and schemas for the plot in Figure 44.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	497	88	102	90
Apache Avro (unframed)	`schema.json`	504	90	103	91
Microsoft Bond (Compact Binary v1)	`schema.bond`	522	116	132	111
Cap'n Proto (Binary Encoding)	`schema.capnp`	640	129	152	120
Cap'n Proto (Packed Encoding)	`schema.capnp`	566	125	135	120
FlatBuffers	`schema.fbs`	668	175	215	164
Protocol Buffers (Binary Wire Format)	`schema.proto`	521	114	133	109
Apache Thrift (Compact Protocol)	`schema.thrift`	521	95	106	96
JSON	-	673	154	195	164
BSON	-	699	160	204	167
CBOR	-	627	147	192	160
FlexBuffers	-	228	214	244	207
MessagePack	-	627	157	192	163
Smile	-	604	168	199	173
UBJSON	-	658	163	198	169

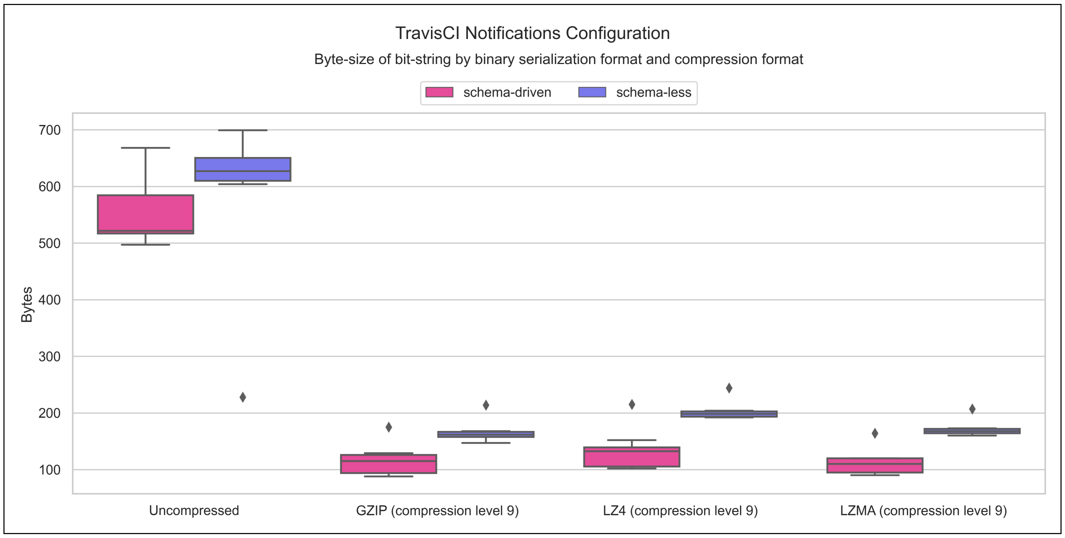

Figure 45: Box plot of the statistical results in Table 40.

between the upper and lower whiskers for schema-less binary serialization specifications are LZ4 and LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that all the considered compression formats are space-efficient in comparison to uncompressed schema-driven and schema-less binary serialization specifications.

6.17 Entry Point Regulation Manifest

Entry Point Regulation (EPR) [52] is a W3C proposal led by Google that defines a manifest that protects websites against cross-site scripting attacks by allowing the developer to mark the areas of the application that can be externally referenced. EPR manifests are used in the web industry. In Figure 46, we demonstrate a **Tier 2 minified $\geq$ 100 $<$ 1000 bytes textual redundant nested** (Tier 2 TRN from Table 2) JSON document that defines an example EPR policy for a fictitious website.

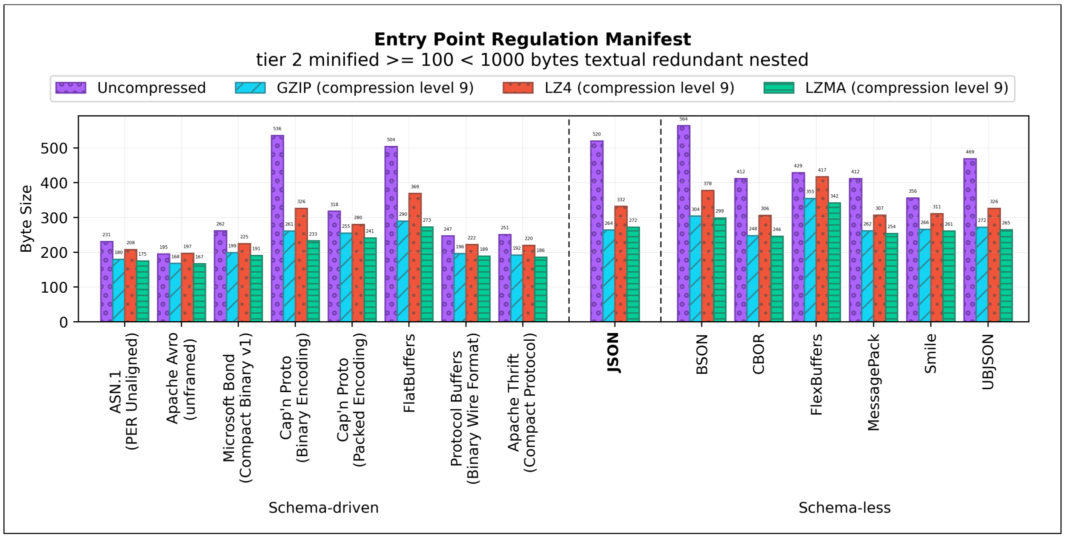

Figure 46: The benchmark results for the Entry Point Regulation Manifest test case listed in Table 4 and Table 5.

The smallest bit-string is produced by Apache Avro [27] (195 bytes) followed by ASN.1 PER Unaligned [57] (231 bytes) and Protocol Buffers [31] (247 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by BSON [42] (564 bytes) followed by Cap'n Proto Binary Encoding [69] (536 bytes) and FlatBuffers [66] (504 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are schema-driven and pointer-based [71]. In comparison to JSON [17] (520 bytes), binary serialization achieves a **2.6x** size reduction in the best case for this input document. However, 2 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: Cap'n Proto Binary Encoding [69] and BSON [42]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 2 TRN document, the best performing schema-driven serialization specification only achieves a **1.8x** size reduction compared to the best performing schema-less serialization specification: Smile [55] (356 bytes). As shown in Table 42, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification only achieves a **1.6x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (318 bytes).

The compression formats listed in subsection 5.5 result in positive gains for all bit-strings. The best performing uncompressed binary serialization specification achieves a **1.3x** size reduction compared to the best performing compression format for JSON: GZIP [16] (264 bytes).

In Figure 47, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications.

In terms of compression, LZMA results in the lower median for both schema-driven and schema-less binary serialization specifications. Additionally, GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to both uncompressed schema-driven and schema-less binary serialization specifications. However, the use of GZIP for schema-less binary serialization specifications exhibit upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA, the compression format with the smaller range between the upper and lower whiskers for schema-less binary serialization specifications is GZIP, and the compression formats with the smaller inter-quartile range for schema-less binary serialization specifications are GZIP and LZMA.

Table 42: A byte-size statistical analysis of the benchmark results shown in Figure 46 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	318	256.5	341	121.2	440.3	420.5	208	64.5
GZIP (compression level 9)	217.6	197.5	122	41.6	284.5	269	107	35.8
LZ4 (compression level 9)	255.9	223.5	172	58.6	340.8	318.5	111	42.1
LZMA (compression level 9)	206.9	190	106	35.1	277.8	263	96	33.2

Table 43: The benchmark raw data results and schemas for the plot in Figure 46.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	231	180	208	175
Apache Avro (unframed)	`schema.json`	195	168	197	167
Microsoft Bond (Compact Binary v1)	`schema.bond`	262	199	225	191
Cap'n Proto (Binary Encoding)	`schema.capnp`	536	261	326	233
Cap'n Proto (Packed Encoding)	`schema.capnp`	318	255	280	241
FlatBuffers	`schema.fbs`	504	290	369	273
Protocol Buffers (Binary Wire Format)	`schema.proto`	247	196	222	189
Apache Thrift (Compact Protocol)	`schema.thrift`	251	192	220	186
JSON	-	520	264	332	272
BSON	-	564	304	378	299
CBOR	-	412	248	306	246
FlexBuffers	-	429	355	417	342
MessagePack	-	412	262	307	254
Smile	-	356	266	311	261
UBJSON	-	469	272	326	265

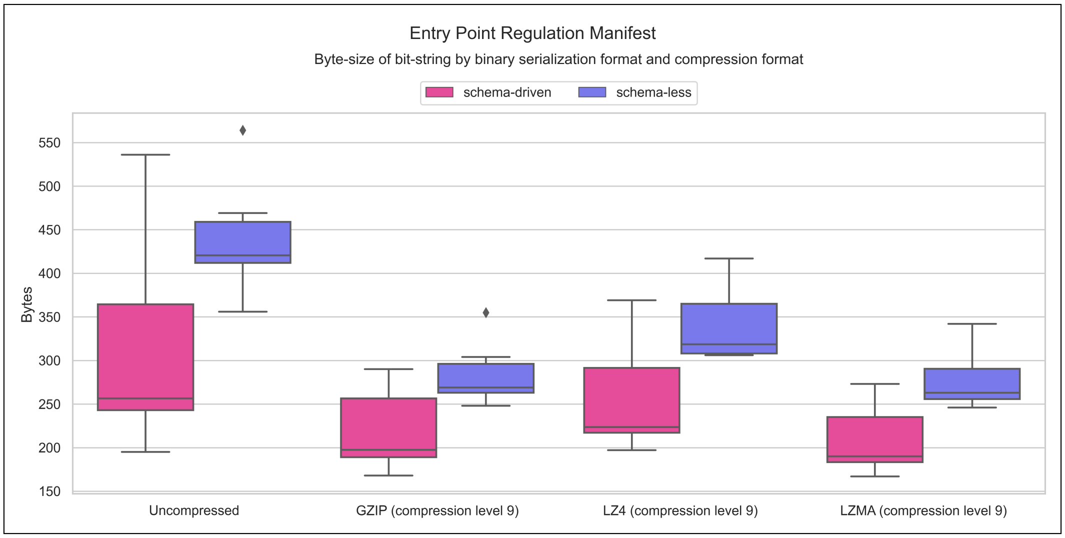

Figure 47: Box plot of the statistical results in Table 42.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that all the considered compression formats are space-efficient in comparison to uncompressed schema-driven and schema-less binary serialization specifications.

6.18 JSON Feed Example Document

JSON Feed [59] is a specification for a syndication JSON format similar to RSS [1] and Atom [44] used in the publishing [89] and media [90] industries. In Figure 48, we demonstrate a **Tier 2 minified $\geq$ 100 $<$ 1000 bytes textual non-redundant flat** (Tier 2 TNF from Table 2) JSON document that consists of a JSON Feed manifest for an example website that contains a single blog entry.

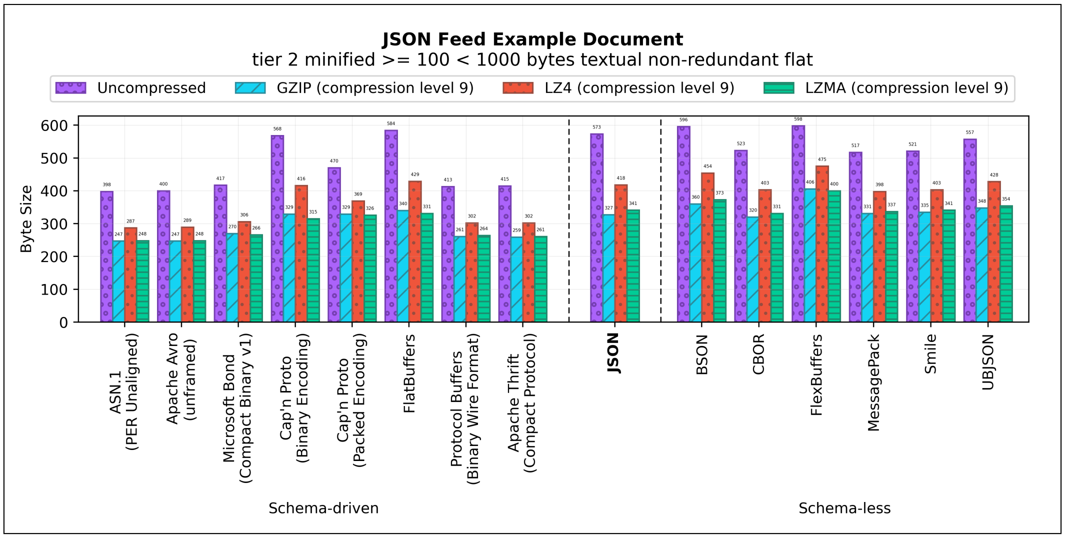

Figure 48: The benchmark results for the JSON Feed Example Document test case listed in Table 4 and Table 5.

The smallest bit-string is produced by ASN.1 PER Unaligned [57] (398 bytes), closely followed by Apache Avro [27] (400 bytes) and Protocol Buffers [31] (413 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by FlexBuffers [67] (598 bytes), closely followed by BSON [42] (596 bytes) and FlatBuffers [66] (584 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are pointer-based [71]. In comparison to JSON [17] (573 bytes), binary serialization only achieves a **1.4x** size reduction in the best case for this input document. Additionally, 3 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: FlatBuffers [66], BSON [42] and FlexBuffers [67]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 2 TNF document, the best performing schema-driven serialization specification only achieves a **1.2x** size reduction compared to the best performing schema-less serialization specification: MessagePack [29] (517 bytes). As shown in Table 44, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification only achieves a **1.1x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (470 bytes).

The compression formats listed in subsection 5.5 result in positive gains for all bit-strings. The best performing compression format for JSON, GZIP [16] (327 bytes), achieves a **1.2x** size reduction compared to the best performing uncompressed binary serialization specification.

[89] `https://micro.blog`

[90] `https://npr.codes/npr-now-supports-json-feed-1c8af29d0ce7`

In Figure 49, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications.

In terms of compression, GZIP and LZMA result in the lower medians for schema-driven binary serialization specifications while GZIP results in the lower median for schema-less binary serialization specifications. Additionally, GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to both uncompressed schema-driven and schema-less binary serialization specifications. However, the use of GZIP for schema-less binary serialization specifications exhibits upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA, and the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-less binary serialization specifications is GZIP.

Table 44: A byte-size statistical analysis of the benchmark results shown in Figure 48 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	458.1	416	186	71.3	552	540	81	34.4
GZIP (compression level 9)	285.3	265.5	93	37.5	350	341.5	86	28.1
LZ4 (compression level 9)	337.5	304	142	54.7	426.8	415.5	77	28.9
LZMA (compression level 9)	282.4	265	83	33.1	356	347.5	69	23.9

Table 45: The benchmark raw data results and schemas for the plot in Figure 48.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	398	247	287	248
Apache Avro (unframed)	`schema.json`	400	247	289	248
Microsoft Bond (Compact Binary v1)	`schema.bond`	417	270	306	266
Cap'n Proto (Binary Encoding)	`schema.capnp`	568	329	416	315
Cap'n Proto (Packed Encoding)	`schema.capnp`	470	329	369	326
FlatBuffers	`schema.fbs`	584	340	429	331
Protocol Buffers (Binary Wire Format)	`schema.proto`	413	261	302	264
Apache Thrift (Compact Protocol)	`schema.thrift`	415	259	302	261
JSON	-	573	327	418	341
BSON	-	596	360	454	373
CBOR	-	523	320	403	331
FlexBuffers	-	598	406	475	400
MessagePack	-	517	331	398	337
Smile	-	521	335	403	341
UBJSON	-	557	348	428	354

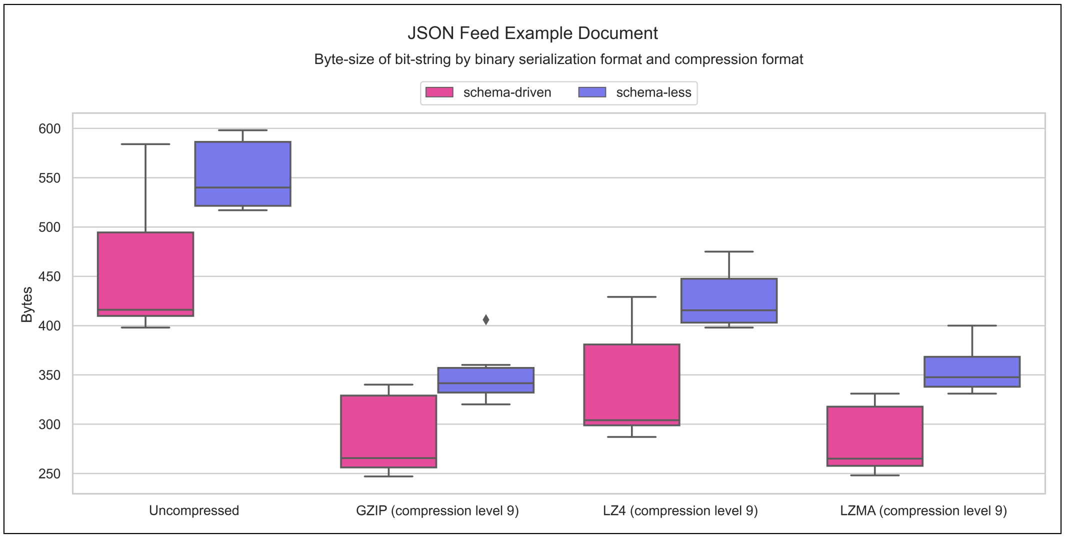

Figure 49: Box plot of the statistical results in Table 44.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that all the considered compression formats are space-efficient in comparison to uncompressed schema-driven and schema-less binary serialization specifications.

6.19 GitHub Workflow Definition

The GitHub [91] software hosting provider has an automation service called GitHub Actions [92] for projects to define custom workflows. GitHub Actions is used primarily by the open-source software industry. In Figure 50, we demonstrate a **Tier 2 minified $\geq$ 100 $<$ 1000 bytes textual non-redundant nested** (Tier 2 TNN from Table 2) JSON document that consists of a simple example workflow definition.

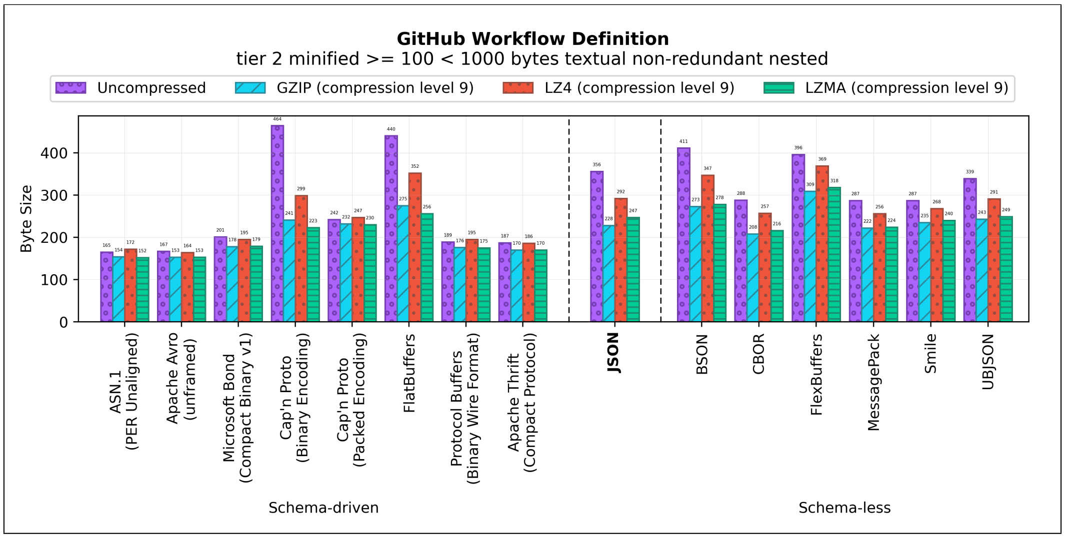

Figure 50: The benchmark results for the GitHub Workflow Definition test case listed in Table 4 and Table 5.

The smallest bit-string is produced by ASN.1 PER Unaligned [57] (165 bytes), closely followed by Apache Avro [27] (167 bytes) and Apache Thrift [60] (187 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by Cap'n Proto Binary Encoding [69] (464 bytes), followed by FlatBuffers [66] (440 bytes) and BSON [42] (411 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are schema-driven and pointer-based [71]. In comparison to JSON [17] (356 bytes), binary serialization achieves a **2.1x** size reduction in the best case for this input document. However, 4 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: Cap'n Proto Binary Encoding [69], FlatBuffers [66], BSON [42] and FlexBuffers [67]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 2 TNN document, the best performing schema-driven serialization specification only achieves a **1.7x** size reduction compared to the best performing schema-less serialization specification: MessagePack [29] and Smile [55] (287 bytes). As shown in Table 46, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification only achieves a **1.4x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (242 bytes).

[91] `https://github.com`

[92] `https://github.com/features/actions`

The compression formats listed in subsection 5.5 result in positive gains for all bit-strings. The best performing uncompressed binary serialization specification achieves a **1.3x** size reduction compared to the best performing compression format for JSON: GZIP [16] (228 bytes).

Table 46: A byte-size statistical analysis of the benchmark results shown in Figure 50 divided by schema-driven and schema-less specifications.

Category	Schema-driven				Schema-less			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	256.9	195	299	115.0	334.7	313.5	124	52.2
GZIP (compression level 9)	197.4	177	122	42.7	248.3	239	101	33.7
LZ4 (compression level 9)	226.3	195	188	63.2	298	279.5	113	44.4
LZMA (compression level 9)	192.3	177	104	36.3	254.2	244.5	102	34.7

Table 47: The benchmark raw data results and schemas for the plot in Figure 50.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	165	154	172	152
Apache Avro (unframed)	`schema.json`	167	153	164	153
Microsoft Bond (Compact Binary v1)	`schema.bond`	201	178	195	179
Cap'n Proto (Binary Encoding)	`schema.capnp`	464	241	299	223
Cap'n Proto (Packed Encoding)	`schema.capnp`	242	232	247	230
FlatBuffers	`schema.fbs`	440	275	352	256
Protocol Buffers (Binary Wire Format)	`schema.proto`	189	176	195	175
Apache Thrift (Compact Protocol)	`schema.thrift`	187	170	186	170
JSON	-	356	228	292	247
BSON	-	411	273	347	278
CBOR	-	288	208	257	216
FlexBuffers	-	396	309	369	318
MessagePack	-	287	222	256	224
Smile	-	287	235	268	240
UBJSON	-	339	243	291	249

In Figure 51, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications. However, their respective quartiles overlap.

In terms of compression, GZIP and LZMA result in the lower medians for schema-driven binary serialization specifications while GZIP results in the lower median for schema-less binary serialization specifications. Additionally, GZIP and LZMA are space-efficient in terms of the median in comparison to uncompressed schema-driven binary serialization specifications and GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to uncompressed schema-less binary serialization specifications. Also, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA, the

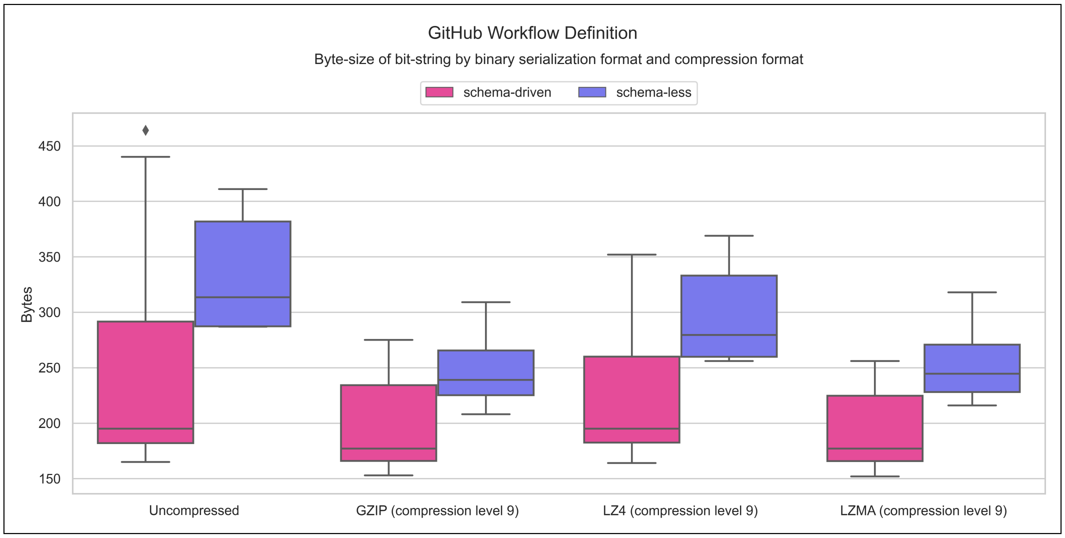

Figure 51: Box plot of the statistical results in Table 46.

compression formats with the smaller range between the upper and lower whiskers for schema-less binary serialization specifications are GZIP and LZMA, and the compression format with the smaller inter-quartile range for schema-less binary serialization specifications is GZIP.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications. GZIP and LZMA are space-efficient in comparison to uncompressed schema-driven binary serialization specifications and all the considered compression formats are space-efficient in comparison to uncompressed schema-less binary serialization specifications.

6.20 GitHub FUNDING Sponsorship Definition (Empty)

The GitHub [93] software hosting provider defines a `FUNDING` [94] file format to declare the funding platforms that an open-source project supports. The `FUNDING` file format is used by the open-source software industry. In Figure 52, we demonstrate a **Tier 2 minified $\geq$ 100 $<$ 1000 bytes boolean redundant flat** (Tier 2 BRF from Table 2) JSON document that consists of a definition that does not declare any supported funding platforms.

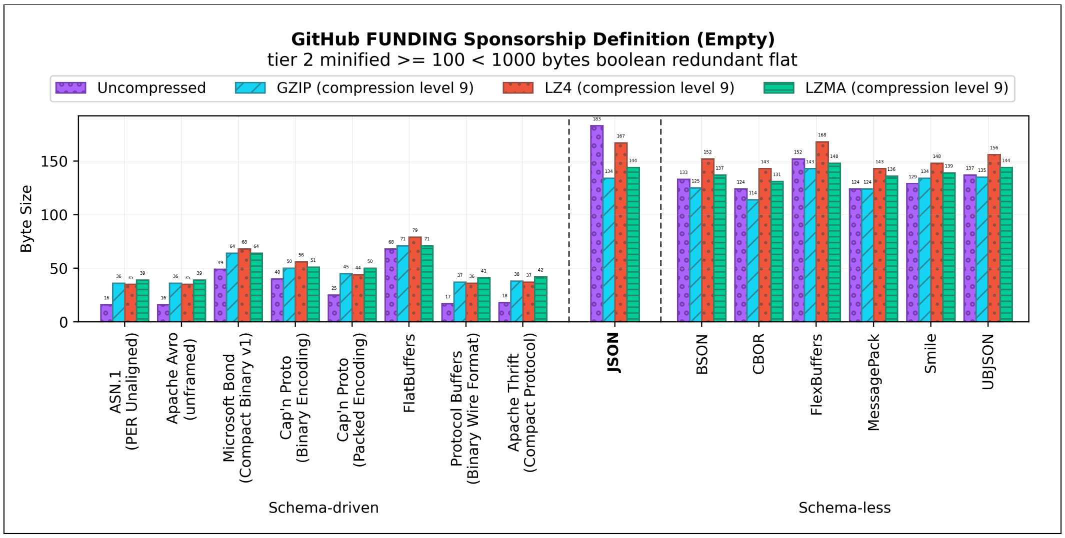

Figure 52: The benchmark results for the GitHub FUNDING Sponsorship Definition (Empty) test case listed in Table 4 and Table 5.

The smallest bit-string is produced by both ASN.1 PER Unaligned [57] and Apache Avro [27] (16 bytes), closely followed by Protocol Buffers [31] (17 bytes) and Apache Thrift [60] (18 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by FlexBuffers [67] (152 bytes), followed by UBJSON [7] (137 bytes) and BSON [42] (133 bytes). The binary serialization specifications that produced the largest bit-strings are schema-less and with the exception of FlexBuffers, they are also sequential [71]. In comparison to JSON [17] (183 bytes), binary serialization achieves a **11.4x** size reduction in the best case for this input document. Similar large size reductions are observed in JSON documents whose content is dominated by *boolean* and *numeric* values. None of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON.

For this Tier 2 BRF document, the best performing schema-driven serialization specification achieves a **7.7x** size reduction compared to the best performing schema-less serialization specification: CBOR [5] and MessagePack [29] (124 bytes). As shown in Table 48, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. The entire selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification only achieves a **1.5x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (25 bytes).

The compression formats listed in subsection 5.5 result in positive gains for the bit-strings produced by JSON [17], BSON [42], CBOR [5], FlexBuffers [67] and UBJSON [7]. The best performing

[93] `https://github.com`

[94] `https://docs.github.com/en/github/administering-a-repository/managing-repository-settings/displaying-a-sponsor-button-in-your-repository`

uncompressed binary serialization specification achieves a **8.3x** size reduction compared to the best performing compression format for JSON: GZIP [16] (134 bytes).

Table 48: A byte-size statistical analysis of the benchmark results shown in Figure 52 divided by schema-driven and schema-less specifications.

Category	Schema-driven				Schema-less			
	Average	Median	Range	Std.dev	Average	Median	Range	Std.dev
Uncompressed	31.1	21.5	52	18.1	133.2	131	28	9.6
GZIP (compression level 9)	47.1	41.5	35	12.8	129.2	129.5	29	9.3
LZ4 (compression level 9)	48.8	40.5	44	16.0	151.7	150	25	8.7
LZMA (compression level 9)	49.6	46	32	11.3	139.2	138	17	5.5

Table 49: The benchmark raw data results and schemas for the plot in Figure 52.

Serialization Format	Schema	Uncompressed	GZIP	LZ4	LZMA
ASN.1 (PER Unaligned)	`schema.asn`	16	36	35	39
Apache Avro (unframed)	`schema.json`	16	36	35	39
Microsoft Bond (Compact Binary v1)	`schema.bond`	49	64	68	64
Cap'n Proto (Binary Encoding)	`schema.capnp`	40	50	56	51
Cap'n Proto (Packed Encoding)	`schema.capnp`	25	45	44	50
FlatBuffers	`schema.fbs`	68	71	79	71
Protocol Buffers (Binary Wire Format)	`schema.proto`	17	37	36	41
Apache Thrift (Compact Protocol)	`schema.thrift`	18	38	37	42
JSON	-	183	134	167	144
BSON	-	133	125	152	137
CBOR	-	124	114	143	131
FlexBuffers	-	152	143	168	148
MessagePack	-	124	124	143	136
Smile	-	129	134	148	139
UBJSON	-	137	135	156	144

In Figure 53, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications.

In terms of compression, LZ4 results in the lower median for schema-driven binary serialization specifications while GZIP results in the lower median for schema-less binary serialization specifications. Compression is not space-efficient in terms of the median in comparison to uncompressed schema-driven binary serialization specifications. However, GZIP is space-efficient in terms of the median in comparison to uncompressed schema-less binary serialization specifications. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for both schema-driven and schema-less binary serialization specifications is LZMA.

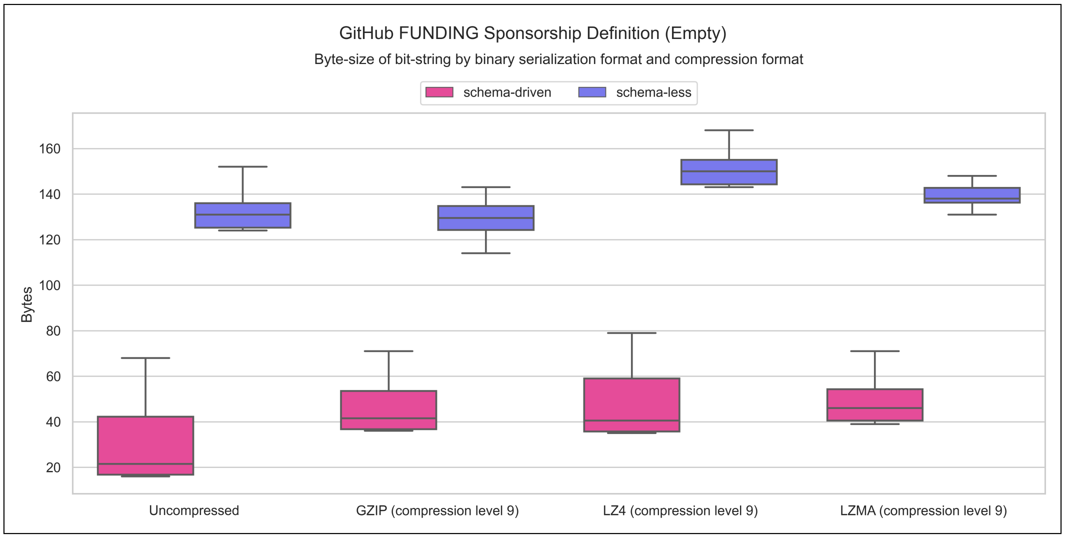

Figure 53: Box plot of the statistical results in Table 48.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications. Compression does not contribute to space-efficiency in comparison to schema-driven binary serialization specifications but GZIP is space-efficient in comparison to schema-less binary serialization specifications.

6.21 ECMAScript Module Loader Definition

esm [95] is an open-source ECMAScript [18] module loader for the Node.js [96] JavaScript runtime that allows developers to use the modern `import` module syntax on older runtime versions. esm is used in the web industry. In Figure 54, we demonstrate a **Tier 2 minified $\geq$ 100 $<$ 1000 bytes boolean non-redundant flat** (Tier 2 BNF from Table 2) JSON document that defines an example esm configuration.

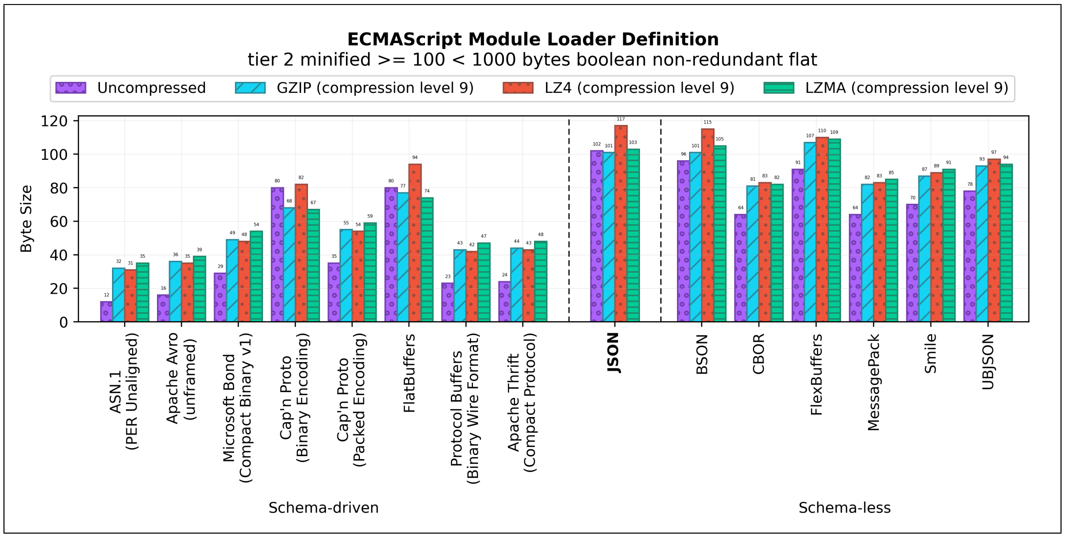

Figure 54: The benchmark results for the ECMAScript Module Loader Definition test case listed in Table 4 and Table 5.

The smallest bit-string is produced by ASN.1 PER Unaligned [57] (12 bytes), followed by Apache Avro [27] (16 bytes) and Protocol Buffers [31] (23 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by BSON [42] (96 bytes), followed by FlexBuffers [67] (91 bytes) and FlatBuffers [66] (80 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are pointer-based [71]. In comparison to JSON [17] (102 bytes), binary serialization achieves a **8.5x** size reduction in the best case for this input document. None of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON.

For this Tier 2 BNF document, the best performing schema-driven serialization specification achieves a **5.3x** size reduction compared to the best performing schema-less serialization specification: CBOR [5] and MessagePack [29] (64 bytes). As shown in Table 50, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification achieves a **2.9x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (35 bytes).

The compression formats listed in subsection 5.5 result in positive gains for the bit-strings produced by Cap'n Proto Binary Encoding [69], FlatBuffers [66] and JSON [17]. The best performing uncompressed binary serialization specification achieves a **8.4x** size reduction compared to the best performing compression format for JSON: GZIP [16] (101 bytes).

[95] `https://github.com/standard-things/esm`
[96] `https://nodejs.org`

In Figure 55, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications.

In terms of compression, LZ4 results in the lower median for schema-driven binary serialization specifications while GZIP results in the lower median for schema-less binary serialization specifications. However, compression is not space-efficient in terms of the median for both schema-driven and schema-less binary serialization specifications. Additionally, the use of LZ4 for schema-driven binary serialization specifications exhibits upper outliers. While compression does not contribute to space-efficiency, it reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA, the compression format with the smaller range between the upper and lower whiskers for schema-less binary serialization specifications is GZIP, and the compression formats with the smaller inter-quartile range for schema-less binary serialization specifications are GZIP and LZMA.

Table 50: A byte-size statistical analysis of the benchmark results shown in Figure 54 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	37.4	26.5	68	25.5	77.2	74	32	12.5
GZIP (compression level 9)	50.5	46.5	45	14.5	91.8	90	26	9.6
LZ4 (compression level 9)	53.6	45.5	63	21.1	96.2	93	32	12.5
LZMA (compression level 9)	52.9	51	39	12.5	94.3	92.5	27	9.8

Table 51: The benchmark raw data results and schemas for the plot in Figure 54.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	12	32	31	35
Apache Avro (unframed)	`schema.json`	16	36	35	39
Microsoft Bond (Compact Binary v1)	`schema.bond`	29	49	48	54
Cap'n Proto (Binary Encoding)	`schema.capnp`	80	68	82	67
Cap'n Proto (Packed Encoding)	`schema.capnp`	35	55	54	59
FlatBuffers	`schema.fbs`	80	77	94	74
Protocol Buffers (Binary Wire Format)	`schema.proto`	23	43	42	47
Apache Thrift (Compact Protocol)	`schema.thrift`	24	44	43	48
JSON	-	102	101	117	103
BSON	-	96	101	115	105
CBOR	-	64	81	83	82
FlexBuffers	-	91	107	110	109
MessagePack	-	64	82	83	85
Smile	-	70	87	89	91
UBJSON	-	78	93	97	94

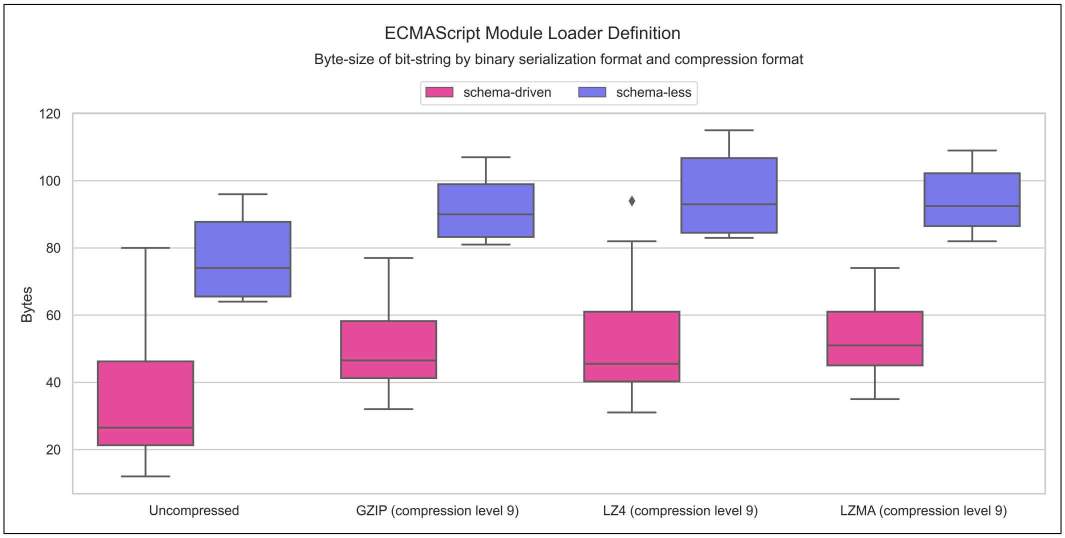

Figure 55: Box plot of the statistical results in Table 50.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that compression does not contribute to space-efficiency in comparison to both uncompressed schema-driven and schema-less binary serialization specifications.

6.22 ESLint Configuration Document

ESLint [97] is a popular open-source extensible linter for the JavaScript [18] programming language used by a wide range of companies in the software development industry such as Google, Salesforce, and Airbnb. In Figure 56, we demonstrate a **Tier 3 minified $\geq$ 1000 bytes numeric redundant flat** (Tier 3 NRF from Table 2) JSON document that defines a browser and Node.js linter configuration that defines general-purposes and *React.js*-specific [98] linting rules.

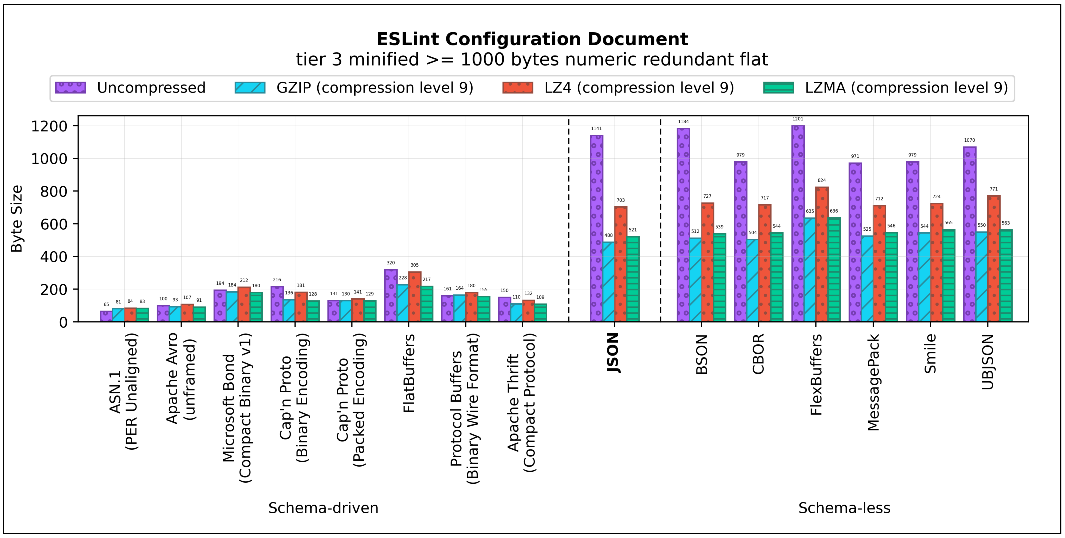

Figure 56: The benchmark results for the ESLint Configuration Document test case listed in Table 4 and Table 5.

The smallest bit-string is produced by ASN.1 PER Unaligned [57] (65 bytes) followed by Apache Avro [27] (100 bytes) and Cap'n Proto Packed Encoding [69] (131 bytes). These serialization specifications are schema-driven and with the exception of Cap'n Proto Packed Encoding, which occupies the third place, they are also sequential [71]. Conversely, the largest bit-string is produced by FlexBuffers [67] (1201 bytes) followed by BSON [42] (1184 bytes) and UBJSON [7] (1070 bytes). The binary serialization specifications that produced the largest bit-strings are schema-less and with the exception of FlexBuffers, they are also sequential [71]. In comparison to JSON [17] (1141 bytes), binary serialization achieves a **17.5x** size reduction in the best case for this input document. Similar large size reductions are observed in JSON documents whose content is dominated by *boolean* and *numeric* values. However, 2 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: BSON [42] and FlexBuffers [67]. These binary serialization specifications are schema-less.

For this Tier 3 NRF document, the best performing schema-driven serialization specification achieves a **14.9x** size reduction compared to the best performing schema-less serialization specification: MessagePack [29] (971 bytes). As shown in Table 52, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. The entire selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification achieves a **2x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (131 bytes).

The compression formats listed in subsection 5.5 result in positive gains for all the bit-strings except the one produced by ASN.1 PER Unaligned [57]. The best performing uncompressed binary serialization specification achieves a **7.5x** size reduction compared to the best performing compression format for JSON: GZIP [16] (488 bytes).

[97] `https://eslint.org`
[98] `https://reactjs.org`

In Figure 57, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications.

In terms of compression, LZMA results in the lower median for schema-driven binary serialization specifications while GZIP results in the lower median for schema-less binary serialization specifications. Additionally, GZIP and LZMA are space-efficient in terms of the median in comparison to uncompressed schema-driven binary serialization specifications and GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to uncompressed schema-less binary serialization specifications. However, the use of LZ4 for schema-driven binary serialization specifications and the use of GZIP, LZ4 and LZMA for schema-less binary serialization specifications exhibit upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers for schema-driven binary serialization specifications is LZ4, and the compression format with the smaller

Table 52: A byte-size statistical analysis of the benchmark results shown in Figure 56 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	167.1	155.5	255	73.4	1064	1024.5	230	96.9
GZIP (compression level 9)	140.8	133	147	46.0	545	534.5	131	43.4
LZ4 (compression level 9)	167.8	160.5	221	65.0	745.8	725.5	112	39.9
LZMA (compression level 9)	136.5	128.5	134	42.6	565.5	554.5	97	33.0

Table 53: The benchmark raw data results and schemas for the plot in Figure 56.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	65	81	84	83
Apache Avro (unframed)	`schema.json`	100	93	107	91
Microsoft Bond (Compact Binary v1)	`schema.bond`	194	184	212	180
Cap'n Proto (Binary Encoding)	`schema.capnp`	216	136	181	128
Cap'n Proto (Packed Encoding)	`schema.capnp`	131	130	141	129
FlatBuffers	`schema.fbs`	320	228	305	217
Protocol Buffers (Binary Wire Format)	`schema.proto`	161	164	180	155
Apache Thrift (Compact Protocol)	`schema.thrift`	150	110	132	109
JSON	-	1141	488	703	521
BSON	-	1184	512	727	539
CBOR	-	979	504	717	544
FlexBuffers	-	1201	635	824	636
MessagePack	-	971	525	712	546
Smile	-	979	544	724	565
UBJSON	-	1070	550	771	563

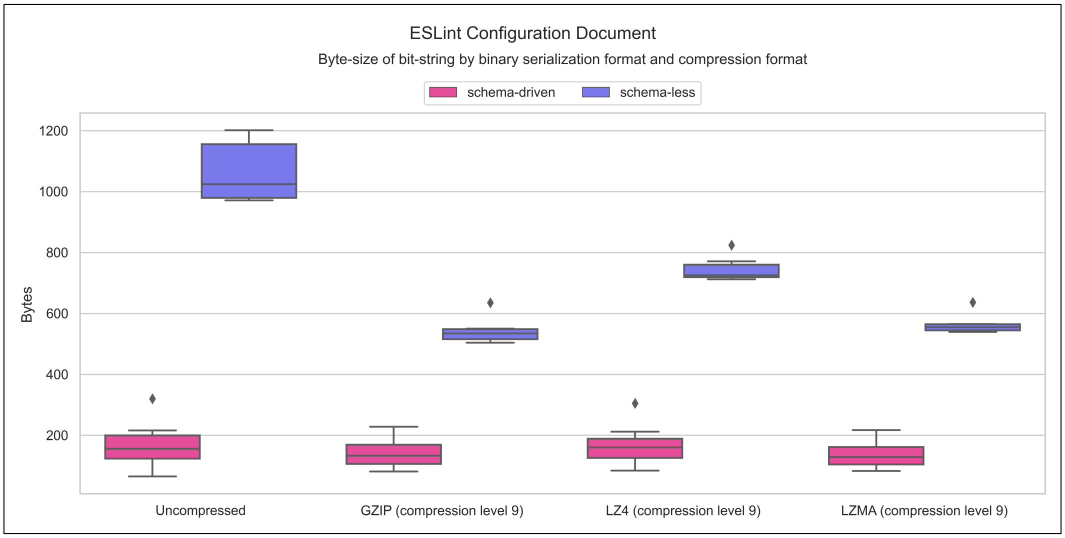

Figure 57: Box plot of the statistical results in Table 52.

range between the upper and lower whiskers and the smaller inter-quartile range for schema-less binary serialization specifications is LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications. GZIP and LZMA are space-efficient in comparison to uncompressed schema-driven binary serialization specifications and all the considered compression formats are space-efficient in comparison to uncompressed schema-less binary serialization specifications.

6.23 NPM Package.json Linter Configuration Manifest

Node.js Package Manager (NPM) [99] is an open-source package manager for Node.js [100], a JavaScript [18] runtime targetted at the web development industry. `npm-package-json-lint` [101] is an open-source tool to enforce a set of configurable rules for a Node.js Package Manager (NPM) [102] configuration manifest. In Figure 58, we demonstrate a **Tier 3 minified $\geq$ 1000 bytes textual redundant flat** (Tier 3 TRF from Table 2) JSON document that consists of an example `npm-package-json-lint` configuration.

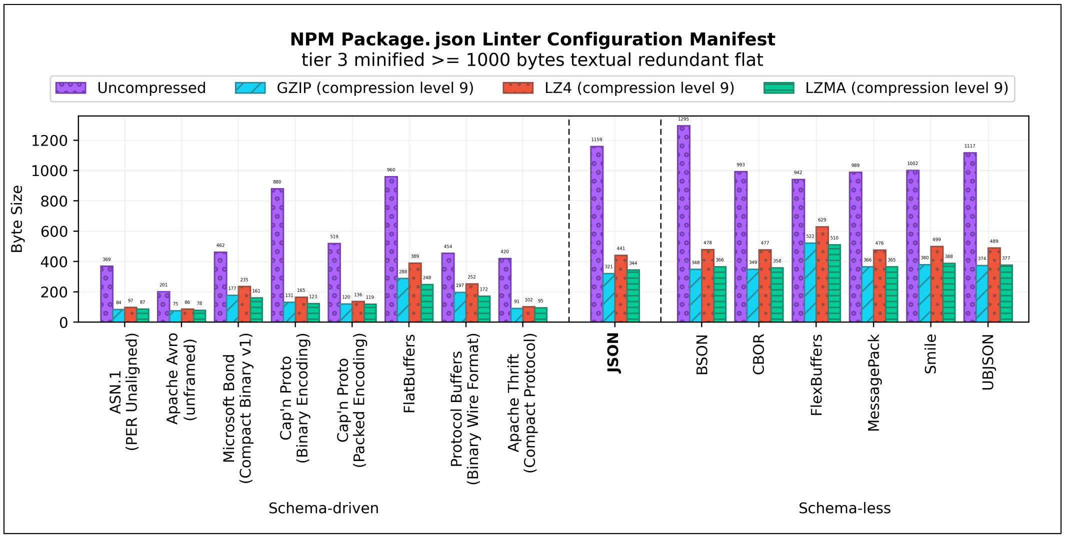

Figure 58: The benchmark results for the NPM Package.json Linter Configuration Manifest test case listed in Table 4 and Table 5.

The smallest bit-string is produced by Apache Avro [27] (201 bytes), followed by ASN.1 PER Unaligned [57] (369 bytes) and Apache Thrift [60] (420 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by BSON [42] (1295 bytes), followed by UBJSON [7] (1117 bytes) and Smile [55] (1002 bytes). The binary serialization specifications that produced the largest bit-strings are schema-less and sequential. In comparison to JSON [17] (1159 bytes), binary serialization achieves a **5.7x** size reduction in the best case for this input document. However, 1 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: the schema-less and sequential BSON [42] binary serialization specification.

For this Tier 3 TRF document, the best performing schema-driven serialization specification achieves a **4.6x** size reduction compared to the best performing schema-less serialization specification: FlexBuffers [67] (942 bytes). As shown in Table 54, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specification FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification achieves a **2.5x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (519 bytes).

[99] `https://www.npmjs.com`
[100] `https://nodejs.org`
[101] `https://npmpackagejsonlint.org/en/`
[102] `https://www.npmjs.com`

The compression formats listed in subsection 5.5 result in positive gains for all bit-strings. The best performing uncompressed binary serialization specification achieves a **1.5x** size reduction compared to the best performing compression format for JSON: GZIP [16] (321 bytes).

Table 54: A byte-size statistical analysis of the benchmark results shown in Figure 58 divided by schema-driven and schema-less specifications.

Category	Schema-driven				Schema-less			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	533.1	458	759	240.9	1056.3	997.5	353	119.2
GZIP (compression level 9)	145.4	125.5	213	67.6	389.8	370	174	60.3
LZ4 (compression level 9)	182.8	150.5	303	97.3	508	483.5	153	54.7
LZMA (compression level 9)	135.4	121	170	52.9	394	371.5	152	52.8

Table 55: The benchmark raw data results and schemas for the plot in Figure 58.

Serialization Format	Schema	Uncompressed	GZIP	LZ4	LZMA
ASN.1 (PER Unaligned)	`schema.asn`	369	84	97	87
Apache Avro (unframed)	`schema.json`	201	75	86	78
Microsoft Bond (Compact Binary v1)	`schema.bond`	462	177	235	161
Cap'n Proto (Binary Encoding)	`schema.capnp`	880	131	165	123
Cap'n Proto (Packed Encoding)	`schema.capnp`	519	120	136	119
FlatBuffers	`schema.fbs`	960	288	389	248
Protocol Buffers (Binary Wire Format)	`schema.proto`	454	197	252	172
Apache Thrift (Compact Protocol)	`schema.thrift`	420	91	102	95
JSON	-	1159	321	441	344
BSON	-	1295	348	478	366
CBOR	-	993	349	477	358
FlexBuffers	-	942	522	629	510
MessagePack	-	989	366	476	365
Smile	-	1002	380	499	388
UBJSON	-	1117	374	489	377

In Figure 59, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications.

In terms of compression, LZMA results in the lower medians for schema-driven binary serialization specifications while GZIP and LZMA result in the lower median for schema-less binary serialization specifications. Additionally, GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to both uncompressed schema-driven and schema-less binary serialization specifications. However, the use of GZIP, LZ4 and LZMA for schema-less binary serialization specifications exhibits upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA.

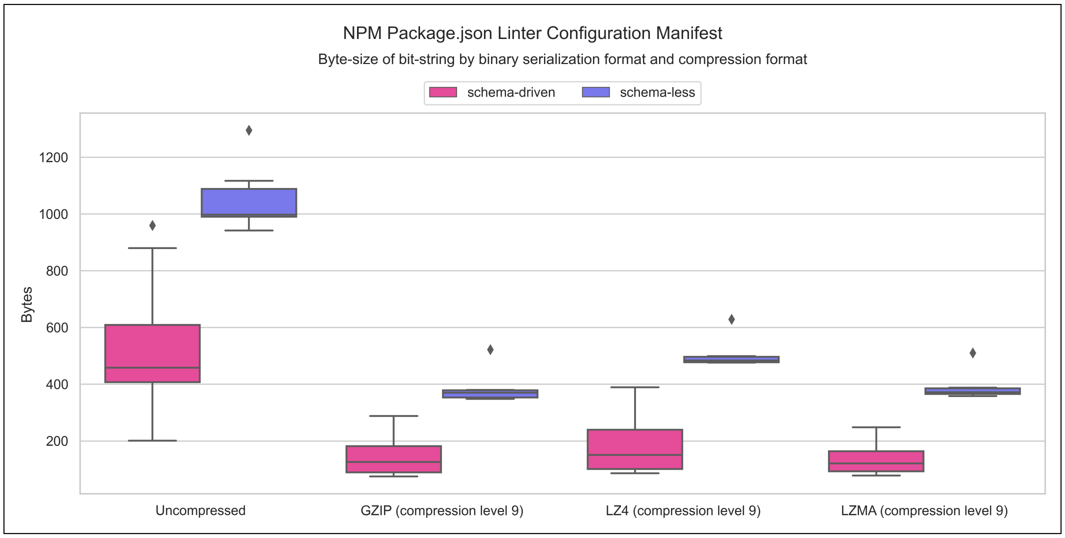

Figure 59: Box plot of the statistical results in Table 54.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that all the considered compression formats are space-efficient in comparison to uncompressed schema-driven and schema-less binary serialization specifications.

6.24 .NET Core Project

The ASP.NET [103] Microsoft web-application framework defined a now-obsolete JSON-based project manifest called `project.json` [104] used in the web industry. In Figure 60, we demonstrate a **Tier 3 minified $\geq$ 1000 bytes textual redundant nested** (Tier 3 TRN from Table 2) JSON document that consists of a detailed an example `project.json` manifest that lists several dependencies.

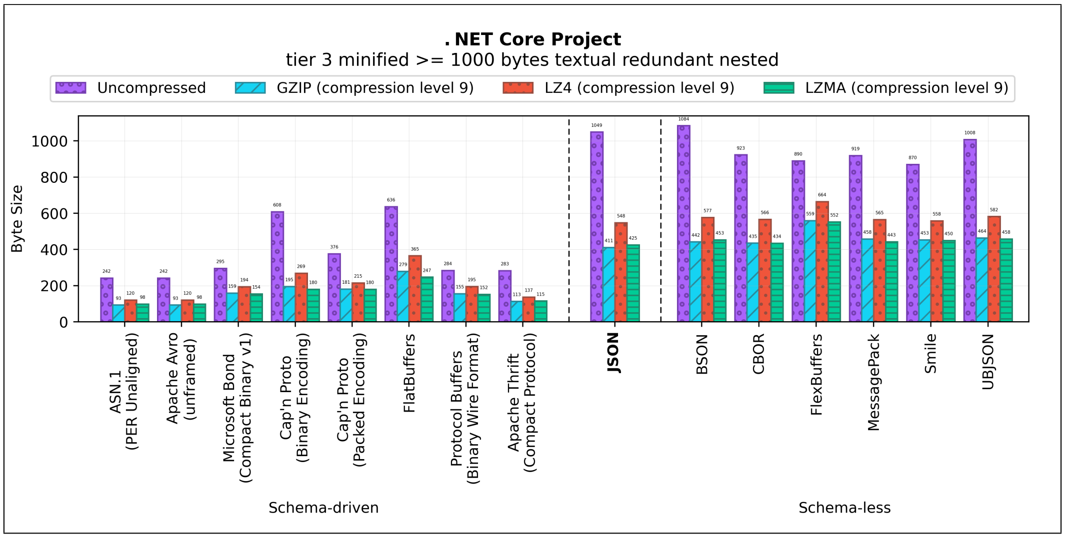

Figure 60: The benchmark results for the .NET Core Project test case listed in Table 4 and Table 5.

The smallest bit-string is produced by both ASN.1 PER Unaligned [57] and Apache Avro [27] (242 bytes), followed by Apache Thrift [60] (283 bytes) and Protocol Buffers [31] (284 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by BSON [42] (1084 bytes), followed by UBJSON [7] (1008 bytes) and CBOR [5] (923 bytes). The binary serialization specifications that produced the largest bit-strings are schema-less and sequential [71]. In comparison to JSON [17] (1049 bytes), binary serialization achieves a **4.3x** size reduction in the best case for this input document. However, 1 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: the schema-less and sequential BSON [42] binary serialization specification.

For this Tier 3 TRN document, the best performing schema-driven serialization specification achieves a **3.5x** size reduction compared to the best performing schema-less serialization specification: Smile [55] (870 bytes). As shown in Table 56, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. The entire selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification only achieves a **1.5x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (376 bytes).

The compression formats listed in subsection 5.5 result in positive gains for all bit-strings. The best performing uncompressed binary serialization specification achieves a **1.6x** size reduction compared to the best performing compression format for JSON: GZIP [16] (411 bytes).

[103] `https://dotnet.microsoft.com/apps/aspnet`

[104] `http://web.archive.org/web/20150322033428/https://github.com/aspnet/Home/wiki/Project.json-file`

In Figure 61, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications.

In terms of compression, LZMA results in the lower median for both schema-driven and schema-less binary serialization specifications. Additionally, GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to both uncompressed schema-driven and schema-less binary serialization specifications. However, the use of GZIP, LZ4 and LZMA for schema-less binary serialization specifications exhibits upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers for schema-driven binary serialization specifications is LZMA, and the compression formats with the smaller inter-quartile range for schema-driven binary serialization specifications are GZIP and LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that all

Table 56: A byte-size statistical analysis of the benchmark results shown in Figure 60 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	370.8	289.5	394	150.3	949	921	214	74.2
GZIP (compression level 9)	158.5	157	186	58.1	468.5	455.5	124	41.6
LZ4 (compression level 9)	201.9	194.5	245	78.3	585.3	571.5	106	36.1
LZMA (compression level 9)	153	153	149	47.2	465	451.5	118	39.6

Table 57: The benchmark raw data results and schemas for the plot in Figure 60.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	242	93	120	98
Apache Avro (unframed)	`schema.json`	242	93	120	98
Microsoft Bond (Compact Binary v1)	`schema.bond`	295	159	194	154
Cap'n Proto (Binary Encoding)	`schema.capnp`	608	195	269	180
Cap'n Proto (Packed Encoding)	`schema.capnp`	376	181	215	180
FlatBuffers	`schema.fbs`	636	279	365	247
Protocol Buffers (Binary Wire Format)	`schema.proto`	284	155	195	152
Apache Thrift (Compact Protocol)	`schema.thrift`	283	113	137	115
JSON	-	1049	411	548	425
BSON	-	1084	442	577	453
CBOR	-	923	435	566	434
FlexBuffers	-	890	559	664	552
MessagePack	-	919	458	565	443
Smile	-	870	453	558	450
UBJSON	-	1008	464	582	458

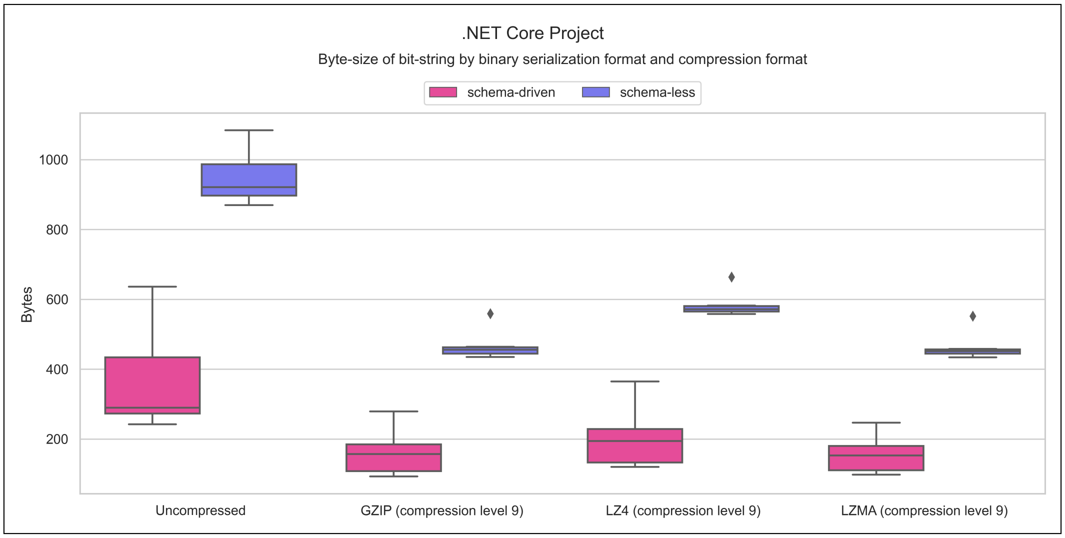

Figure 61: Box plot of the statistical results in Table 56.

the considered compression formats are space-efficient in comparison to uncompressed schema-driven and schema-less binary serialization specifications.

6.25 NPM Package.json Example Manifest

Node.js Package Manager (NPM) [105] is an open-source package manager for Node.js [106], a JavaScript [18] runtime targetted at the web development industry. A package that is published to NPM is declared using a JSON file called `package.json` [107]. In Figure 62, we demonstrate a **Tier 3 minified $\geq$ 1000 bytes textual non-redundant flat** (Tier 3 TNF from Table 2) JSON document that consists of a `package.json` manifest that declares a particular version of the Grunt.js [108] task runner.

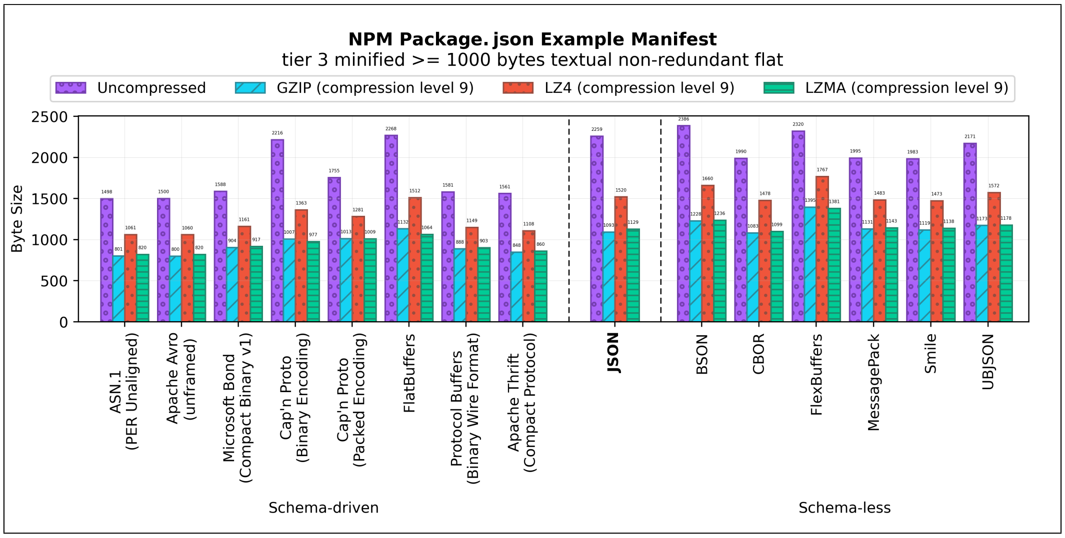

Figure 62: The benchmark results for the NPM Package.json Example Manifest test case listed in Table 4 and Table 5.

The smallest bit-string is produced by ASN.1 PER Unaligned [57] (1498 bytes), closely followed by Apache Avro [27] (1500 bytes) and Apache Thrift [60] (1561 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by BSON [42] (2386 bytes), followed by FlexBuffers [67] (2320 bytes) and FlatBuffers [66] (2268 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are pointer-based [71]. In comparison to JSON [17] (2259 bytes), binary serialization only achieves a **1.5x** size reduction in the best case for this input document. Additionally, 3 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: FlatBuffers [66], BSON [42] and FlexBuffers [67]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 3 TNF document, the best performing schema-driven serialization specification only achieves a **1.3x** size reduction compared to the best performing schema-less serialization specification: Smile [55] (1983 bytes). As shown in Table 58, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification only achieves a **1.1x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (1755 bytes).

[105] `https://www.npmjs.com`
[106] `https://nodejs.org`
[107] `https://docs.npmjs.com/cli/v6/configuring-npm/package-json`
[108] `https://gruntjs.com`

The compression formats listed in subsection 5.5 result in positive gains for all bit-strings. The best performing compression format for JSON, GZIP [16] (1093 bytes), achieves a **1.3x** size reduction compared to the best performing uncompressed binary serialization specification.

Table 58: A byte-size statistical analysis of the benchmark results shown in Figure 62 divided by schema-driven and schema-less specifications.

Category	Schema-driven				Schema-less			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	1745.9	1584.5	770	296.2	2140.8	2083	403	164.3
GZIP (compression level 9)	924.1	896	332	109.6	1188.2	1152	312	103.0
LZ4 (compression level 9)	1211.9	1155	452	150.4	1572.2	1527.5	294	109.8
LZMA (compression level 9)	921.3	910	244	83.5	1195.8	1160.5	282	92.9

Table 59: The benchmark raw data results and schemas for the plot in Figure 62.

Serialization Format	Schema	Uncompressed	GZIP	LZ4	LZMA
ASN.1 (PER Unaligned)	`schema.asn`	1498	801	1061	820
Apache Avro (unframed)	`schema.json`	1500	800	1060	820
Microsoft Bond (Compact Binary v1)	`schema.bond`	1588	904	1161	917
Cap'n Proto (Binary Encoding)	`schema.capnp`	2216	1007	1363	977
Cap'n Proto (Packed Encoding)	`schema.capnp`	1755	1013	1281	1009
FlatBuffers	`schema.fbs`	2268	1132	1512	1064
Protocol Buffers (Binary Wire Format)	`schema.proto`	1581	888	1149	903
Apache Thrift (Compact Protocol)	`schema.thrift`	1561	848	1108	860
JSON	-	2259	1093	1520	1129
BSON	-	2386	1228	1660	1236
CBOR	-	1990	1083	1478	1099
FlexBuffers	-	2320	1395	1767	1381
MessagePack	-	1995	1131	1483	1143
Smile	-	1983	1119	1473	1138
UBJSON	-	2171	1173	1572	1178

In Figure 63, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications.

In terms of compression, GZIP results in the lower median for both schema-driven and schema-less binary serialization specifications. Additionally, GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to both uncompressed schema-driven and schema-less binary serialization specifications. However, the use of GZIP and LZMA for schema-less binary serialization specifications exhibits upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA, and the compression formats with the smaller range between the upper and

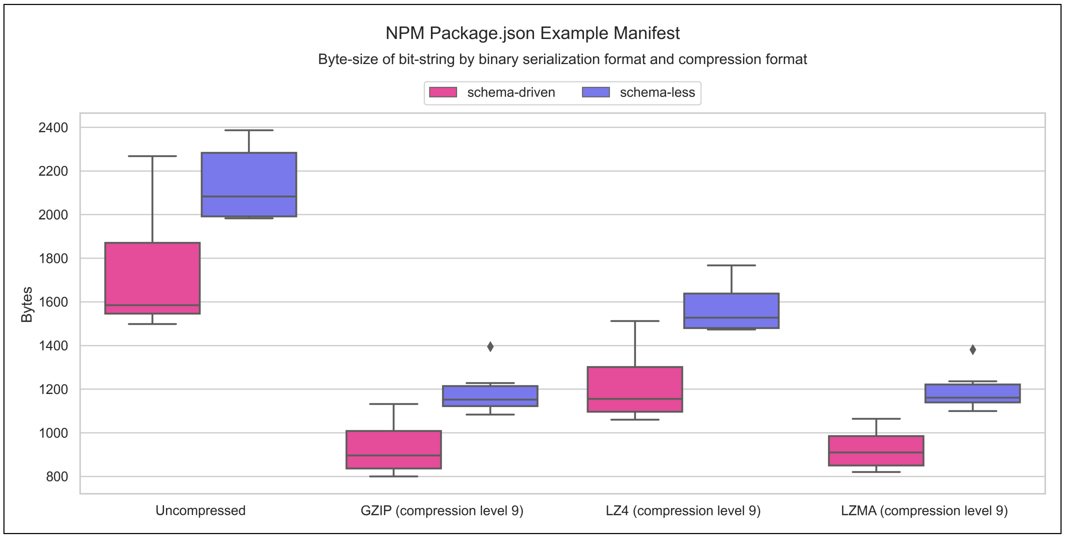

Figure 63: Box plot of the statistical results in Table 58.

lower whiskers and the smaller inter-quartile range for schema-less binary serialization specifications are GZIP and LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that all the considered compression formats are space-efficient in comparison to uncompressed schema-driven and schema-less binary serialization specifications.

6.26 JSON Resume Example

JSON Resume [109] is a community-driven proposal for a JSON-based file format that declares and renders themable resumes used in the recruitment industry. In Figure 64, we demonstrate a **Tier 3 minified $\geq$ 1000 bytes textual non-redundant nested** (Tier 3 TNN from Table 2) JSON document that consists of a detailed example resume for a fictitious software programmer.

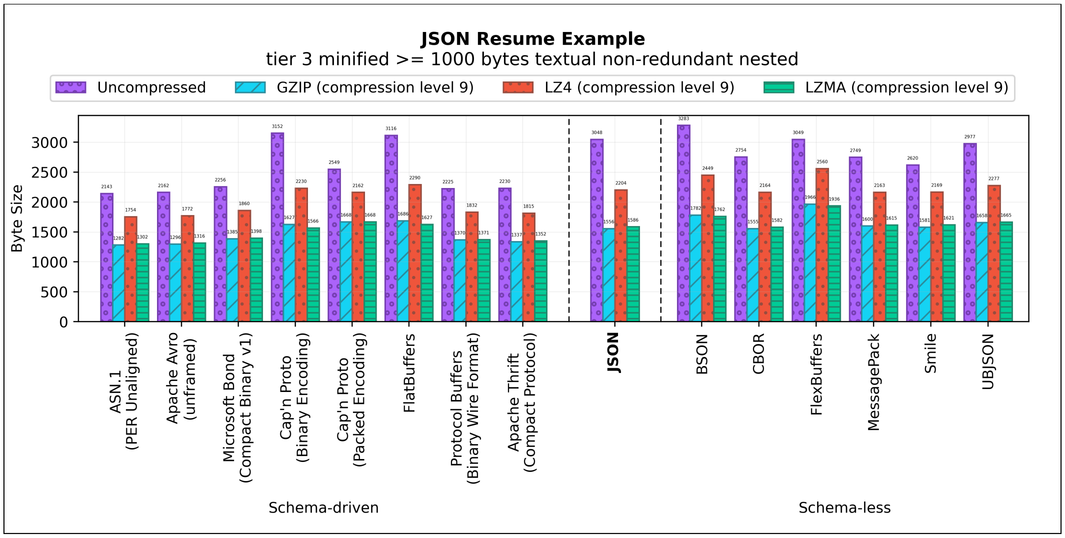

Figure 64: The benchmark results for the JSON Resume Example test case listed in Table 4 and Table 5.

The smallest bit-string is produced by ASN.1 PER Unaligned [57] (2143 bytes), closely followed by Apache Avro [27] (2162 bytes) and Protocol Buffers [31] (2225 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by BSON [42] (3283 bytes), followed by Cap'n Proto Binary Encoding [69] (3152 bytes) and FlatBuffers [66] (3116 bytes). With the exception of BSON, the binary serialization specifications that produced the largest bit-strings are schema-driven and pointer-based [71]. In comparison to JSON [17] (3048 bytes), binary serialization only achieves a **1.4x** size reduction in the best case for this input document. Additionally, 4 out of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON: Cap'n Proto Binary Encoding [69], FlatBuffers [66], BSON [42] and FlexBuffers [67]. These binary serialization specifications are either schema-less or schema-driven and pointer-based.

For this Tier 3 TNN document, the best performing schema-driven serialization specification only achieves a **1.2x** size reduction compared to the best performing schema-less serialization specification: Smile [55] (2620 bytes). As shown in Table 60, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. With the exception of the pointer-based binary serialization specifications Cap'n Proto Binary Encoding [69] and FlatBuffers [66], the selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification only achieves a **1.1x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (2549 bytes).

The compression formats listed in subsection 5.5 result in positive gains for all bit-strings. The best performing compression format for JSON, GZIP [16] (1556 bytes), achieves a **1.3x** size reduction compared to the best performing uncompressed binary serialization specification.

[109] `https://jsonresume.org`

In Figure 65, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-less binary serialization specifications is smaller than the range between the upper and lower whiskers and the inter-quartile range of uncompressed schema-driven binary serialization specifications.

In terms of compression, GZIP results in the lower median for both schema-driven and schema-less binary serialization specifications. Additionally, GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to both uncompressed schema-driven and schema-less binary serialization specifications. However, the use of LZMA for schema-less binary serialization specifications exhibits upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller range between the upper and lower whiskers and the smaller inter-quartile range for both schema-driven and schema-less binary serialization specifications is LZMA.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications and that all

Table 60: A byte-size statistical analysis of the benchmark results shown in Figure 64 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	2479.1	2243	1009	395.8	2905.3	2865.5	663	222.5
GZIP (compression level 9)	1456.4	1377.5	404	161.8	1690.3	1629	411	143.7
LZ4 (compression level 9)	1964.4	1846	536	208.5	2297	2223	397	155.3
LZMA (compression level 9)	1450	1384.5	366	137.3	1696.8	1643	354	121.2

Table 61: The benchmark raw data results and schemas for the plot in Figure 64.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	2143	1282	1754	1302
Apache Avro (unframed)	`schema.json`	2162	1296	1772	1316
Microsoft Bond (Compact Binary v1)	`schema.bond`	2256	1385	1860	1398
Cap'n Proto (Binary Encoding)	`schema.capnp`	3152	1627	2230	1566
Cap'n Proto (Packed Encoding)	`schema.capnp`	2549	1668	2162	1668
FlatBuffers	`schema.fbs`	3116	1686	2290	1627
Protocol Buffers (Binary Wire Format)	`schema.proto`	2225	1370	1832	1371
Apache Thrift (Compact Protocol)	`schema.thrift`	2230	1337	1815	1352
JSON	-	3048	1556	2204	1586
BSON	-	3283	1782	2449	1762
CBOR	-	2754	1555	2164	1582
FlexBuffers	-	3049	1966	2560	1936
MessagePack	-	2749	1600	2163	1615
Smile	-	2620	1581	2169	1621
UBJSON	-	2977	1658	2277	1665

Figure 65: Box plot of the statistical results in Table 60.

the considered compression formats are space-efficient in comparison to uncompressed schema-driven and schema-less binary serialization specifications.

6.27 Nightwatch.js Test Framework Configuration

Nightwatch.js [110] is an open-source browser automation solution used in the software testing industry. In Figure 66, we demonstrate a **Tier 3 minified $\geq$ 1000 bytes boolean redundant flat** (Tier 3 BRF from Table 2) JSON document that consists of a Nightwatch.js configuration file that defines a set of general-purpose WebDriver [61] and Selenium [111] options.

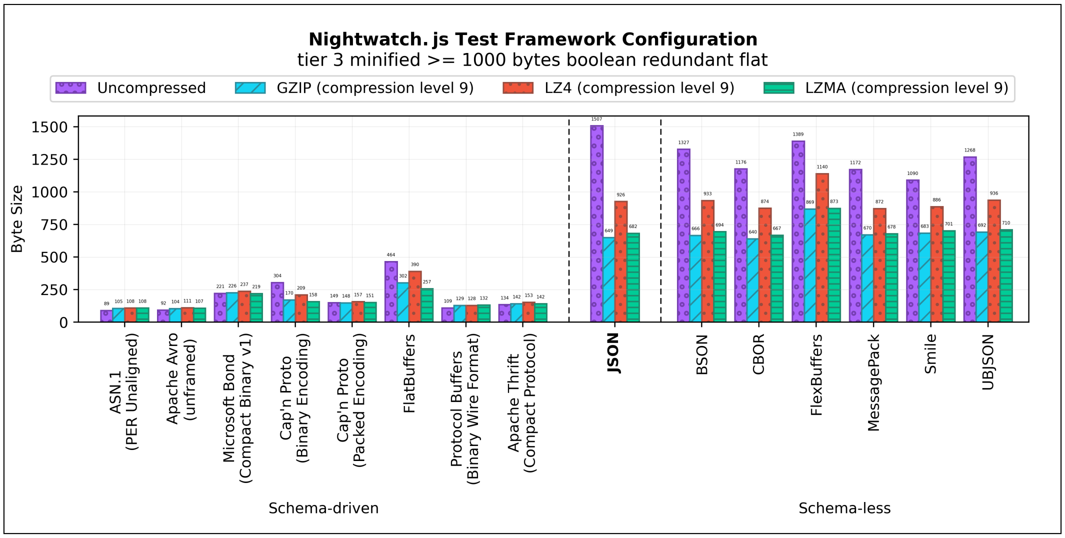

Figure 66: The benchmark results for the Nightwatch.js Test Framework Configuration test case listed in Table 4 and Table 5.

The smallest bit-string is produced by ASN.1 PER Unaligned [57] (89 bytes), followed by Apache Avro [27] (92 bytes) and Protocol Buffers [31] (109 bytes). The binary serialization specifications that produced the smallest bit-strings are schema-driven and sequential [71]. Conversely, the largest bit-string is produced by FlexBuffers [67] (1389 bytes), followed by BSON [42] (1327 bytes) and UBJSON [7] (1268 bytes). The binary serialization specifications that produced the largest bit-strings are schema-less and with the exception of FlexBuffers, they are also sequential [71]. In comparison to JSON [17] (1507 bytes), binary serialization achieves a **16.9x** size reduction in the best case for this input document. Similar large size reductions are observed in JSON documents whose content is dominated by *boolean* and *numeric* values. None of the 14 JSON-compatible binary serialization specifications listed in Table 6 and Table 7 result in bit-strings that are larger than JSON.

For this Tier 3 BRF document, the best performing schema-driven serialization specification achieves a **12.2x** size reduction compared to the best performing schema-less serialization specification: Smile [55] (1090 bytes). As shown in Table 62, uncompressed schema-driven specifications provide smaller *average* and *median* bit-strings than uncompressed schema-less specifications. However, as highlighted by the *range* and *standard deviation*, uncompressed schema-driven specifications exhibit higher size reduction variability depending on the expressiveness of the schema language (i.e. how the language constructs allow you to model the data) and the size optimizations devised by its authors. The entire selection of schema-driven serialization specifications listed in Table 6 produce bit-strings that are equal to or smaller than their schema-less counterparts listed in Table 7. The best performing sequential serialization specification only achieves a **1.6x** size reduction compared to the best performing pointer-based serialization specification: Cap'n Proto Packed Encoding [69] (149 bytes).

The compression formats listed in subsection 5.5 result in positive gains for all bit-strings except the ones produced by ASN.1 PER Unaligned [57], Apache Avro [27], Microsoft Bond [43], Protocol Buffers [31] and Apache Thrift [60]. The best performing uncompressed binary serialization specification achieves a **7.2x** size reduction compared to the best performing compression format for JSON: GZIP [16] (649 bytes).

[110] `https://nightwatchjs.org`
[111] `https://www.selenium.dev`

In Figure 67, we observe the medians for uncompressed schema-driven binary serialization specifications to be smaller in comparison to uncompressed schema-less binary serialization specifications. The range between the upper and lower whiskers of uncompressed schema-driven binary serialization specifications is smaller than the range between the upper and lower whiskers of uncompressed schema-less binary serialization specifications. However, the inter-quartile range of both both uncompressed schema-driven and schema-less binary serialization specifications is similar.

In terms of compression, GZIP and LZMA result in the lower medians for schema-driven binary serialization specifications while GZIP results in the lower median for schema-less binary serialization specifications. Compression is not space-efficient in terms of the median in comparison to uncompressed schema-driven binary serialization specifications. However, GZIP, LZ4 and LZMA are space-efficient in terms of the median in comparison to uncompressed schema-less binary serialization specifications. Additionally, the use of GZIP, LZ4 and LZMA for both schema-driven binary serialization specifications and schema-less binary serialization specifications exhibits upper outliers. Nevertheless, compression reduces the range between the upper and lower whiskers and inter-quartile range for both schema-driven and schema-less binary serialization specifications. In particular, the compression format with the smaller inter-quartile range for schema-driven binary serialization specifications is LZMA, the compression format with the smaller range between the upper and lower

Table 62: A byte-size statistical analysis of the benchmark results shown in Figure 66 divided by schema-driven and schema-less specifications.

Category	**Schema-driven**				**Schema-less**			
	Average	**Median**	**Range**	**Std.dev**	**Average**	**Median**	**Range**	**Std.dev**
Uncompressed	195.3	141.5	375	122.5	1237	1222	299	101.4
GZIP (compression level 9)	165.8	145	198	63.2	703.3	676.5	229	75.8
LZ4 (compression level 9)	186.6	155	282	87.9	940.2	909.5	268	93.1
LZMA (compression level 9)	159.3	146.5	150	49.5	720.5	697.5	206	69.7

Table 63: The benchmark raw data results and schemas for the plot in Figure 66.

Serialization Format	**Schema**	**Uncompressed**	**GZIP**	**LZ4**	**LZMA**
ASN.1 (PER Unaligned)	`schema.asn`	89	105	108	108
Apache Avro (unframed)	`schema.json`	92	104	111	107
Microsoft Bond (Compact Binary v1)	`schema.bond`	221	226	237	219
Cap'n Proto (Binary Encoding)	`schema.capnp`	304	170	209	158
Cap'n Proto (Packed Encoding)	`schema.capnp`	149	148	157	151
FlatBuffers	`schema.fbs`	464	302	390	257
Protocol Buffers (Binary Wire Format)	`schema.proto`	109	129	128	132
Apache Thrift (Compact Protocol)	`schema.thrift`	134	142	153	142
JSON	-	1507	649	926	682
BSON	-	1327	666	933	694
CBOR	-	1176	640	874	667
FlexBuffers	-	1389	869	1140	873
MessagePack	-	1172	670	872	678
Smile	-	1090	683	886	701
UBJSON	-	1268	692	936	710

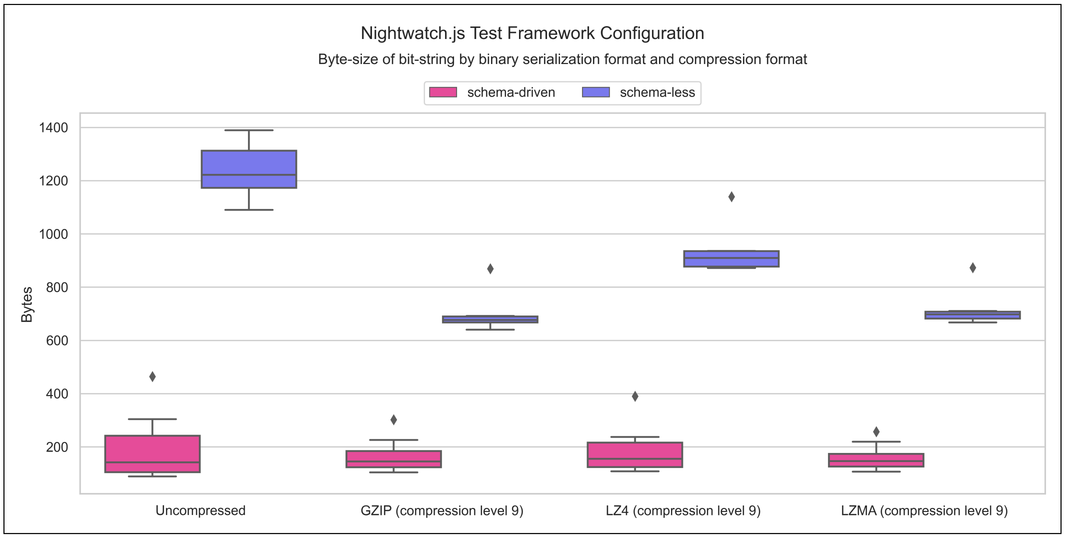

Figure 67: Box plot of the statistical results in Table 62.

whiskers for schema-less binary serialization specifications is LZMA, and the compression format with the smaller inter-quartile range for schema-less binary serialization specifications is GZIP.

Overall, we conclude that uncompressed schema-driven binary serialization specifications are space-efficient in comparison to uncompressed schema-less binary serialization specifications. Compression does not contribute to space-efficiency in comparison to schema-driven binary serialization specifications but all the considered compression formats are space-efficient in comparison to schema-less binary serialization specifications.

7 Reproducibility

To encourage reproducibility for the benchmarking datasets, we follow every reproducibility level introduced by [33]. We found that the implemented benchmark software matches the definition of Level 3, the highest-level of reproducibility, as justified in the following sections.

- **Automation.** The benchmark software, from the generation of the serialized bit-strings to the generation of the plots using Matplotlib [112], is automated through a GNU Make [113] declarative and parallelizable build definition.
- **Testing.** The POSIX shell and Python scripts distributed with the benchmark are automatically linted using the *shellcheck* [114] and *flake8* [115] open-source tools, respectively. The serialization and deserialization procedures of the benchmark are automatically tested as explained in subsection 5.4.
- **Supported Environments.** The benchmark software is ensured to work on the macOS (Intel processors) and GNU/Linux operating systems. We do not make any effort to support the Microsoft Windows operating system, but we expect the benchmark software to run on an *msys2* [116] or *Windows Subsystem for Linux* [117] environment with minor changes at most. The benchmark is exclusively concerned with the byte-size of the bit-strings produced by the binary serialization specifications. Therefore, the CPU, memory, and network bandwidth characteristics of the test machine do not affect the results of the benchmark. No further

[112] `https://matplotlib.org`
[113] `https://www.gnu.org/software/make/`
[114] `https://www.shellcheck.net`
[115] `https://flake8.pycqa.org/`
[116] `https://www.msys2.org`
[117] `https://docs.microsoft.com/en-us/windows/wsl/`

conditions apart from the exact software versions of the dependencies included and required by the project are necessary to replicate the results.

- **Documentation and Readability.** The `README` file [118] in the repository contains precise instructions for running the benchmark locally and generate the data files and plots. The project documentation includes a detailed list of the system dependencies that are required to succesfully execute every part of the benchmark and a detailed list of the required binary serialization specifications, implementations, versions, and encodings. The benchmark source code is compact and easy to understand and navigate due to the declarative rule definition nature of GNU Make.
- **DOI.** The version of the benchmark software described in this study is archived with a DOI [70]. The DOI includes the source code for reproducing the benchmark and the presented results.
- **Dependencies.** The benchmark software is implemented using established open-source software with the exception of the ASN-1Step [119] command-line tool, which is a proprietary implementation of ASN.1 [56] distributed by OSS Nokalva with a 30 days free trial. Every binary serialization specification implementation used in the benchmark with the exception of ASN-1Step is pinned to its specific version to ensure reproducibility. As explained in the online documentation, the benchmark software expects the ASN-1Step command-line tool version 10.0.2 to be installed and globally-accessible in the system in order to benchmark the ASN.1 PER Unaligned [57] binary serialization specification.
- **Version Control.** The benchmark repository utilises the *git* [120] version control system and its publicly hosted on GitHub [121] as recommended by [46].
- **Continuous Integration.** The GitHub repository hosting the benchmark software is setup with the GitHub Actions [122] continuous integration provided to re-run the benchmark automatically on new commits using a GNU/Linux Ubuntu 20.04 LTS cloud worker. This process prevents changes to the benchmark software from introducing regressions and new software errors. We make use of this process to validate GitHub internal and external pull requests before merging them into the trunk.
- **Availability.** The benchmark software and results are publicly available and governed by the *Apache License 2.0* [123] open-source software license. The results of the benchmark are also published as a website hosted at `https://www.jviotti.com/binary-json-size-benchmark/` using the GitHub Pages free static hosting provider. The website provides direct links to the JSON [17] documents being encoded by the benchmark and direct links to the schema definitions used in every case. Both the JSON documents and the schema definitions are hosted in the benchmark GitHub repository to ensure their availability even if the original sources do not exist anymore.
- **Continuity.** We plan to continue extending the benchmark software in the future to test new versions of the current selection of binary serialization specifications and to include new JSON-compatible binary serialization specifications. We hope for this project to become a collaborative effort to measure the space-efficiency of every new JSON-compatible serialization specifications and We are comitted to accepting open-source contributions.

8 Conclusions

8.1 Q1: How do JSON-compatible schema-less binary serialization specifications compare to JSON in terms of space-efficiency?

Table 64 demonstrates that the median size reduction of the selection of schema-less binary serialization specifications listed in Table 7 is 9.1% and the average size reductions of the selection of

[118] `https://github.com/jviotti/binary-json-size-benchmark#running-locally`
[119] `https://www.oss.com/asn1/products/asn-1step/asn-1step.html`
[120] `https://git-scm.com`
[121] `https://github.com/jviotti/binary-json-size-benchmark`
[122] `https://github.com/features/actions`
[123] `https://www.apache.org/licenses/LICENSE-2.0.html`

schema-less binary serialization specifications listed in Table 7 is 8.2% for the selection of input data set described in Table 4 and Table 5. In comparison to JSON [17], FlexBuffers [67] and BSON [42] often result in larger bit-strings. In comparison to JSON, both CBOR [5] and MessagePack [29] are strictly superior in terms of space-efficiency. In both cases, the median and average size reductions ranged between 22.4% and 22.8% for the selection of input data. Compared to the other schema-less binary serialization specifications, MessagePack [29] tends to provide the best size reductions in the *Tier 1 Minified < 100 bytes* and *Tier 2 Minified ≥ 100 < 1000 bytes* categories while Smile [55] tends to provide the best size reductions for *Tier 3 Minified ≥ 1000 bytes* JSON documents. As a notable positive exception shown in Figure 68, FlexBuffers [67] outperforms the rest of the schema-less binary serialization specifications in two cases: the *Tier 2 Minified ≥ 100 < 1000 bytes, textual, redundant, and flat* (Tier 2 TRF) JSON document from subsection 6.16 (C) and the *Tier 3 Minified ≥ 1000 bytes, textual, redundant, and flat* (Tier 3 TRF) JSON document from subsection 6.23 given its automatic string deduplication features [71]. Figure 68 shows that CBOR [5] and MessagePack [29] tend to outperform the other schema-less binary serialization specifications in terms of space-efficiency while producing stable results with no noticeable outliers. In comparison, BSON [42] (A and B) and FlexBuffers [67] (C and D) produce noticeable outliers at both sides of the spectrum while remaining less space-efficient than the rest of the schema-less binary serialization specifications in most cases. Like BSON [42] (B), Smile [55] produces a negative outlier (E) for the Tier 2 NRN case from subsection 6.13.

Summary. There exists schema-less binary serialization specifications that are space-efficient in comparison to JSON [17]. Based on our findings, we conclude that using MessagePack [29] on *Tier 1 Minified < 100 bytes* and *Tier 2 Minified ≥ 100 < 1000 bytes* JSON documents, Smile [55] on *Tier 3 Minified ≥ 1000 bytes* JSON documents, and FlexBuffers [67] on JSON documents with high-redundancy of *textual* values increases space-efficiency.

Table 64: A summary of the size reduction results in comparison to JSON [17] of the selection of schema-less binary serialization specifications listed in Table 7 against the input data listed in Table 4 and Table 5. See Figure 68 for a visual representation of this data.

Serialization Specification	Size Reductions in Comparison To JSON					Negative Cases
	Maximum	Minimum	Range	Median	Average	
BSON	34.1%	-140.0%	174.1	-7.7%	-16.8%	21 / 27 (77.7%)
CBOR	43.2%	6.8%	36.3	22.5%	22.4%	0 / 27 (0%)
FlexBuffers	66.1%	-65.3%	131.4	-4.1%	-4.9%	16 / 27 (59.2%)
MessagePack	43.2%	6.8%	36.3	22.7%	22.8%	0 / 27 (0%)
Smile	31.8%	-20.0%	51.8	14.2%	15.5%	2 / 27 (7.4%)
UBJSON	34.1%	-9.5%	43.6	7.1%	9.9%	1 / 27 (3.7%)
Averages	**42.1%**	**-36.8%**	**78.9**	**9.1%**	**8.2%**	**24.6%**

8.2 Q2: How do JSON-compatible schema-driven binary serialization specifications compare to JSON and JSON-compatible schema-less binary serialization specifications in terms of space-efficiency?

As illustrated in Table 65, the median size reduction of the selection of schema-driven binary serialization specifications listed in Table 6 is more than five times higher that the schema-less binary serialization specification size reductions listed in Table 64 and the average size reduction of the selection of schema-driven binary serialization specifications listed in Table 6 is more than five times higher that the schema-less binary serialization specification size reductions listed in Table 64 for the given input data described in Table 4 and Table 5. FlatBuffers [66] and Cap'n Proto [69] (unpacked) tend to be less space-efficient than the selection of schema-less binary serialization specifications and are surpassed by the rest of the schema-driven binary serialization specifications listed in Table 6 for most cases. On the other side, ASN.1 PER Unaligned [57] and Apache Avro (unframed) [27] are the most space-efficient schema-driven binary serialization specifications in 23 out of the 27 cases listed in Table 4. Most of the schema-driven binary serialization specifications we considered are strictly superior to JSON [17] and to the schema-less binary serialization specifications listed in Table 7 in terms of message size with a common exception: ASN.1 PER Unaligned [57], Apache Avro (unframed) [27], Microsoft Bond (Compact Binary v1) [43], Protocol Buffers [31], Apache Thrift

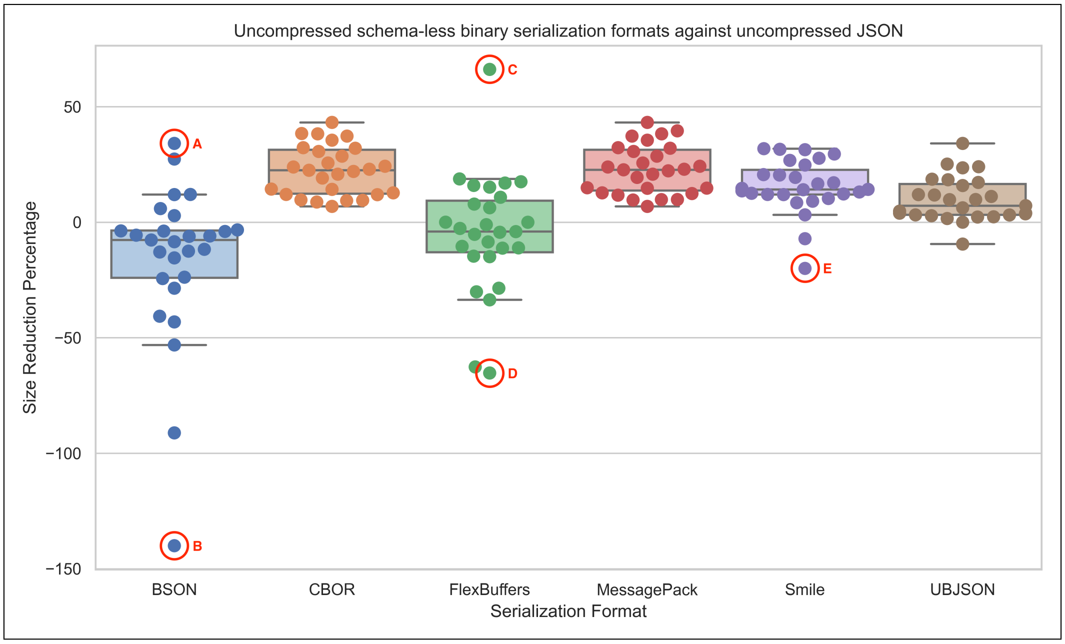

Figure 68: A box plot that demonstrates the size reduction (in percentages) of the selection of schema-less binary serialization specifications listed in Table 7 in comparison to uncompressed JSON [17] given the input data listed in Table 4 and Table 5.

(Compact Protocol) [60], and Cap'n Proto [69] (packed) perform less space-efficiently than JSON [17] and the schema-less binay serialization specifications listed in Table 7 in the *Tier 2 Minified* $\geq$ *100* $<$ *1000 bytes, numeric, redundant, and nested* (Tier 2 NRN) GeoJSON [11] document from subsection 6.13. With the exception of ASN.1 PER Unaligned [57] and Cap'n Proto Packed Encoding [69], the selection of schema-driven binary serialization specifications result in negative outliers for the Tier 2 NRN case as shown in Figure 69 (A, B, C, D, E and F). Compared to the other JSON documents from the input data set, this JSON document consists of highly nested arrays and almost no object keys. Leaving that exception aside, we found that in general, the schema-driven binary serialization specifications listed Table 6 provide the highest space-efficiency improvements in comparison to JSON [17] on *boolean* documents and tend to provide the least space-efficient improvements on *textual* JSON documents. Figure 69 shows that schema-driven binary serialization specifications, in particular ASN.1 PER Unaligned [57], Apache Avro [27], Protocol Buffers [31] and Apache Thrift [60], result in high size reductions in comparison to JSON. However, every considered schema-driven binary serialization specification results in at least one negative space-efficiency exception.

Summary. The schema-driven binary serialization specifications listed in Table 6 tend to be more space-efficient than the schema-less binary serialization specifications listed in Table 7 and JSON [17] in most cases. Based on our findings, we conclude that ASN.1 PER Unaligned [57] and Apache Avro (unpacked) [27] are space-efficient in comparison to schema-less binary serialization specifications in almost all cases as they provide over 70% median size reductions and over 65% average size reductions in comparison to JSON [17].

8.3 Q3: How do JSON-compatible sequential binary serialization specifications compare to JSON-compatible pointer-based binary serialization specifications in terms of space-efficiency?

In terms of the schema-less binary serialization specifications listed in Table 7, Table 64 illustrates that in comparison to JSON [17], FlexBuffers [67] results in negative median and average size reductions, a characteristic only otherwise applicable to BSON [42]. Leaving BSON aside, FlexBuffers only results in more space-efficient messages than a strict subset of the sequential schema-less binary serialization specifications in three cases: *Tier 1 TRN* (subsection 6.6), *Tier 2 TRN* (subsection 6.17)

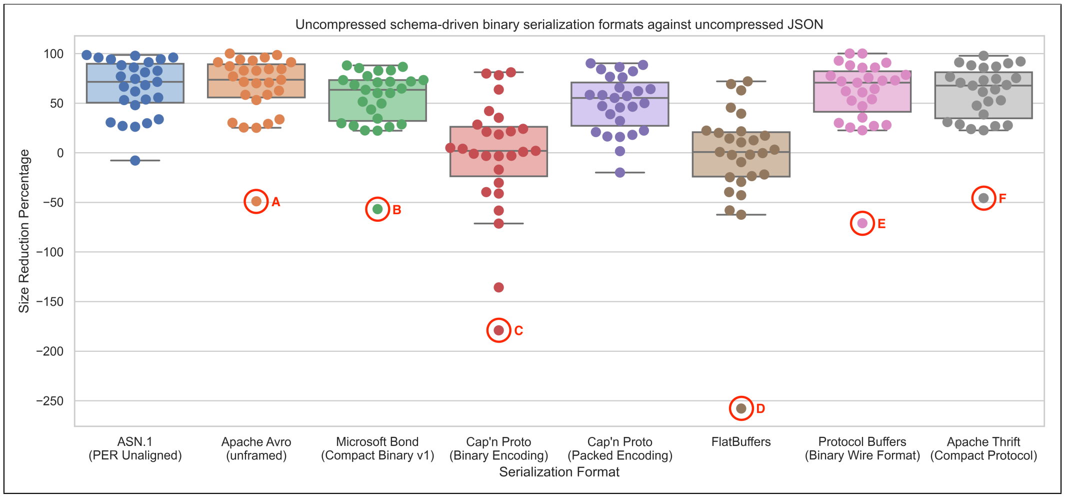

Figure 69: A box plot that demonstrates the size reduction (in percentages) of the selection of schema-driven binary serialization specifications listed in Table 6 in comparison to uncompressed JSON [17] given the input data listed in Table 4 and Table 5.

and *Tier 3 TRN* (subsection 6.24). Furthemore, FlexBuffers [67] is comparatively more space-efficient that all the other schema-less binary serialization specifications listed in Table 7 for the *Tier 2 TRF* JSON document from subsection 6.16 and the *Tier 3 TRF* JSON document from subsection 6.23. However, as explained in subsection 8.1, this is due to FlexBuffers automatic string deduplication feature, which is orthogonal to whether a binary serialization specification is sequential or pointer-based.

We refer to the schema-driven binary serialization specifications listed in Table 6. Table 65 illustrates that the selection of sequential schema-driven binary serialization specifications are strictly superior to FlatBuffers [66] in terms of space reductions. Similarly, Cap'n Proto [69] (unpacked) provides a more space-efficient bit-string than a single sequential schema-driven binary serialization specification, Microsoft Bond [43] (Compact Binary v1), in a single case: *Tier 2 BRF* (subsection 6.20). However,

Table 65: A summary of the size reduction results in comparison to JSON [17] of the selection of schema-driven binary serialization specifications listed in Table 6 against the input data listed in Table 4 and Table 5. See Figure 69 for a visual representation of this data.

Serialization Specification	**Size Reductions in Comparison To JSON**					**Negative Cases**
	Maximum	**Minimum**	**Range**	**Median**	**Average**	
ASN.1 (PER Unaligned)	98.5%	-7.9%	106.4	71.4%	65.7%	1 / 27 (3.7%)
Apache Avro (unframed)	100%	-48.9%	148.9	73.5%	65.7%	1 / 27 (3.7%)
Microsoft Bond (Compact Binary v1)	88%	-56.8%	144.8	63.4%	54%	1 / 27 (3.7%)
Cap'n Proto	81.1%	-179.1%	260.1	1.9%	-2.9%	12 / 27 (44.4%)
Cap'n Proto (packed)	90.1%	-20%	110.1	55.2%	49.6%	1 / 27 (3.7%)
FlatBuffers	72%	-257.9%	329.8	0.7%	-6.1%	13 / 27 (48.1%)
Protocol Buffers	100%	-71.1%	171.1	70.6%	59.3%	1 / 27 (3.7%)
Apache Thrift (Compact Protocol)	97.7%	-45.8%	143.5	67.6%	58.1%	1 / 27 (3.7%)
Averages	**90.9%**	**-85.9%**	**176.9**	**50.6%**	**42.9%**	**14.3%**

Cap'n Proto [69] (packed) results in more space-efficient messages than a strict subset of the sequential schema-driven binary serialization specifications in six cases: *Tier 1 NNF* (subsection 6.3), *Tier 1 BRF* (subsection 6.9), *Tier 2 NRN* (subsection 6.13), *Tier 2 BRF* (subsection 6.20), *Tier 3 NRF* (subsection 6.22), and *Tier 3 BRF* (subsection 6.27); but never surpasses the entire set of sequential schema-driven binary serialization specifications for any JSON document from the input data set listed in Table 4 and Table 5.

Summary. Based on our findings, sequential binary serialization specifications are typically more space-efficient than pointer-based binary serialization specifications, independent of whether they are schema-less or schema-driven.

8.4 Q4: How does compressed JSON compares to uncompressed and compressed JSON-compatible binary serialization specifications?

8.4.1 Data Compression

We found that data compression tends to yield negative results on *Tier 1 Minified* $<$ *100 bytes* JSON documents. As an extreme, LZMA resulted in a negative 171.4% size reduction for subsection 6.3. The entire selection of data compression formats produced negative results for all the *Tier 1 Minified* $<$ *100 bytes* JSON documents we considered except for subsection 6.10, for which LZ4 produced a negative result but GZIP [16] and LZMA resulted in a 8.2% and 6.1% reduction, respectively, and subsection 6.6, for which all data compression formats produced positive results ranging from 10.4% in the case of LZ4 to 16.7% in the case of GZIP [16]. Leaving *Tier 1 Minified* $<$ *100 bytes* JSON documents aside, all the data compression formats we selected offered better average and median compression ratios on *textual* JSON documents as seen in Table 66. Out of the selection of data compression formats, GZIP [16] performed better in terms of the average and median size reduction in all *Tier 2 Minified* $\geq$ *100* $<$ *1000 bytes* and *Tier 3 Minified* $\geq$ *1000 bytes* categories.

8.4.2 Schema-less Binary Serialization Specifications

Table 67 summarizes the size reductions provided by schema-less binary serialization specifications in comparison to compressed JSON [17]. Leaving BSON [42] and FlexBuffers [67] aside, schema-less binary serialization specifications typically provide space-efficient results in *Tier 1 Minified* $<$ *100 bytes* JSON documents, as these usually resulted in negative compression ratios. However, compressed JSON provides space-efficient results in 15 out of the 27 listed in Figure 11. In comparison to

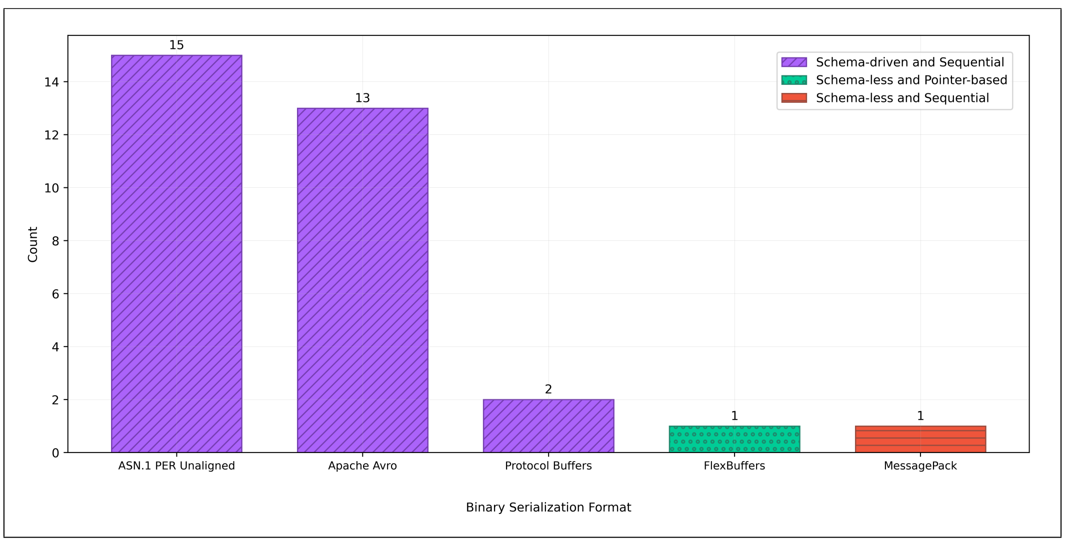

Figure 70: The binary serialization specifications that resulted in the highest size reductions for each JSON [17] document for the input data listed in Table 4 and Table 5, broken down by type. Schema-driven sequential binary serialization specifications, in particular ASN.1 PER Unaligned [57] and Apache Avro [27], resulted in the highest size reductions in most cases.

compressed JSON, no schema-less binary serialization provides both a positive median and average size reduction. As shown in Figure 71, the selection of schema-less binary serialization specifications listed in Table 7, with the exception of FlexBuffers [67], result in negative outliers for the Tier 2 TRF case from subsection 6.16 (A, B, C, D, E).

As summarized in Table 68, compressing the bit-strings produced by schema-less binary serialization specifications results in 22 out 90 instances that are space-efficient in comparison to compressed JSON on *Tier 2 Minified* $\geq$ *100* $<$ *1000 bytes* and *Tier 3 Minified* $\geq$ *1000 bytes* JSON documents but reduces the advantages that uncompressed schema-less binary serialization specifications have over compressed JSON on *Tier 1 Minified* $<$ *100 bytes* JSON documents. In comparison to compressed JSON, compressed CBOR [5] is strictly equal or superior than the rest of the compressed schema-less binary serialization specifications in all but a single case: *Tier 1 NRN* from subsection 6.2, providing the highest median (8.8%) and highest average (8.1%) size reductions. As a notable outlier shown in Figure 72, best-case compressed BSON [42] results in a negative size reduction of 44% in comparison to compressed JSON [17] for the Tier 2 NRN case from subsection 6.13.

8.4.3 Schema-driven Binary Serialization Specifications

As shown in Table 69, schema-driven binary serialization specifications provide positive median and average size reductions in comparison to compressed JSON [17]. However, schema-driven binary serialization specifications tend to produce negative results in comparison to compressed JSON mostly on *Tier 2 Minified* $\geq$ *100* $<$ *1000 bytes textual* (22 out of 32 cases) and *Tier 3 Minified* $\geq$ *1000 bytes textual* (25 out of 32) JSON documents. Even when taking compression into account, both ASN.1 PER Unaligned [57] and Apache Avro (unpacked) [27] continue to provide over 70% median size reductions and almost 40% average size reductions. As shown in Figure 73, the entire selection of schema-driven binary serialization specifications listed in Table 6 results in negative outliers for the Tier 2 TRF case from subsection 6.16 (A, B, C, D, E, G and H) and the Tier 2 NRN case from subsection 6.13 (F).

Compressing the bit-strings produced by schema-driven binary serialization specifications shows that compressed *sequential* schema-driven binary serialization specifications are strictly superior than compressed JSON [17] as shown in Table 70. On the higher end, both ASN.1 PER Unaligned [57] and Apache Avro [27] provide median and average size reductions of over 50% in comparison

Table 66: The average and median size reduction of using the selection of data compression formats on the *Tier 2 Minified* $\geq$ *100* $<$ *1000 bytes* and *Tier 3 Minified* $\geq$ *1000 bytes* input JSON documents. GZIP [16] resulted in higher compression ratios for all categories.

Compression Format	**Numeric**		**Textual**		**Boolean**	
	Average	*Median*	*Average*	*Median*	*Average*	*Median*
GZIP (compression level 9)	39%	33.3%	54%	49.2%	28%	26.8%
LZ4 (compression level 9)	21%	19.5%	40%	32.7%	20%	8.7%
LZMA (compression level 9)	38%	32.8%	52%	48%	25%	21.3%

Table 67: A summary of the size reduction results in comparison to the best case scenarios of compressed JSON [17] given the compression formats listed in Table 9 of the selection of schema-less binary serialization specifications listed in Table 7 against the input data listed in Table 4 and Table 5. See Figure 71 for a visual representation of this data.

Serialization Specification	**Size Reductions in Comparison To Compressed JSON**					**Negative Cases**
	Maximum	**Minimum**	**Range**	**Median**	**Average**	
BSON	50.0%	-353.9%	403.9	-40.8%	-76.9%	22 / 27 (81.4%)
CBOR	69.7%	-307.1%	376.8	7.5%	-26.8%	13 / 27 (48.1%)
FlexBuffers	45.5%	-193.5%	238.9	-48.1%	-50.8%	20 / 27 (74%)
MessagePack	69.7%	-307.1%	376.8	7.5%	-26.2%	13 / 27 (48.1%)
Smile	54.5%	-292.2%	346.8	-5%	-31.7%	14 / 27 (51.8%)
UBJSON	60.6%	-327.3%	387.9	-16.3%	-43.6%	15 / 27 (55.5%)
Averages	**58.3%**	**-296.9%**	**355.2**	**-15.9%**	**-42.7%**	**59.8%**

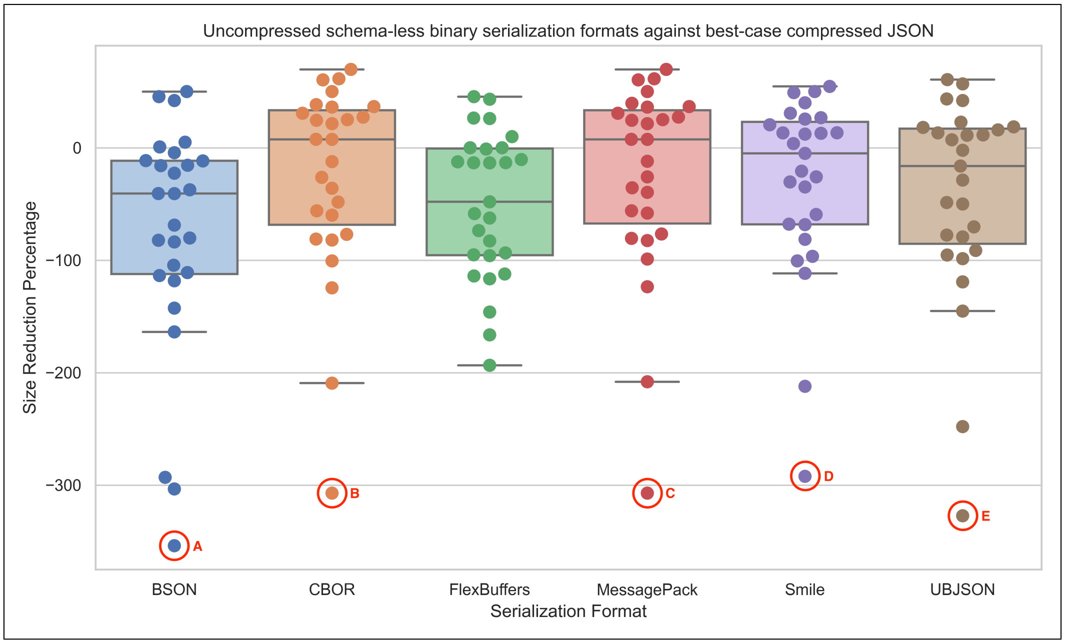

Figure 71: A box plot that demonstrates the size reduction (in percentages) of the selection of schema-less binary serialization specifications listed in Table 7 in comparison to the best-case compressed JSON [17] given the compression formats listed in Table 9 and the input data listed in Table 4 and Table 5.

to compressed JSON, with a minimum size reduction of over 11% in the Tier 2 NRN case from subsection 6.13 for which all the schema-driven binary serialization specifications previously resulted in negative size reductions in comparison to uncompressed JSON. As a notable exception shown in Figure 74, best-case compressed FlatBuffers [66] results in a negative size reduction of 68.1% (A) in comparison to compressed JSON [17] for the Tier 2 NRN case.

Summary. In comparison to compressed JSON, both compressed and uncompressed schema-less binary serialization specifications result in negative median and average size reductions. However,

Table 68: A summary of the size reduction results of the best case scenarios of compressed schema-less binary serialization specifications listed in Table 7 in comparison to the best case scenarios of compressed JSON [17] given the compression formats listed in Table 9 and the the input data listed in Table 4 and Table 5. See Figure 72 for a visual representation of this data.

Serialization Specification	**Size Reductions in Comparison To Compressed JSON**					**Negative Cases**
	Maximum	**Minimum**	**Range**	**Median**	**Average**	
Compressed BSON	8%	-44%	52	-10.1%	-11%	23 / 27 (85.1%)
Compressed CBOR	24.5%	-8.7%	33.3	8.8%	8.1%	4 / 27 (14.8%)
Compressed FlexBuffers	0%	-58.9%	58.9	-24.4%	-23.8%	27 / 27 (100%)
Compressed MessagePack	24.5%	-13.7%	38.2	7.5%	5.9%	10 / 27 (37%)
Compressed Smile	13.9%	-18.4%	32.2	-1.6%	-1.6%	14 / 27 (51.8%)
Compressed UBJSON	13.6%	-16.5%	30.1	-0.7%	-1.9%	15 / 27 (55.5%)
Averages	**14.1%**	**-26.7%**	**40.8**	**-3.4%**	**-4.1%**	**57.3%**

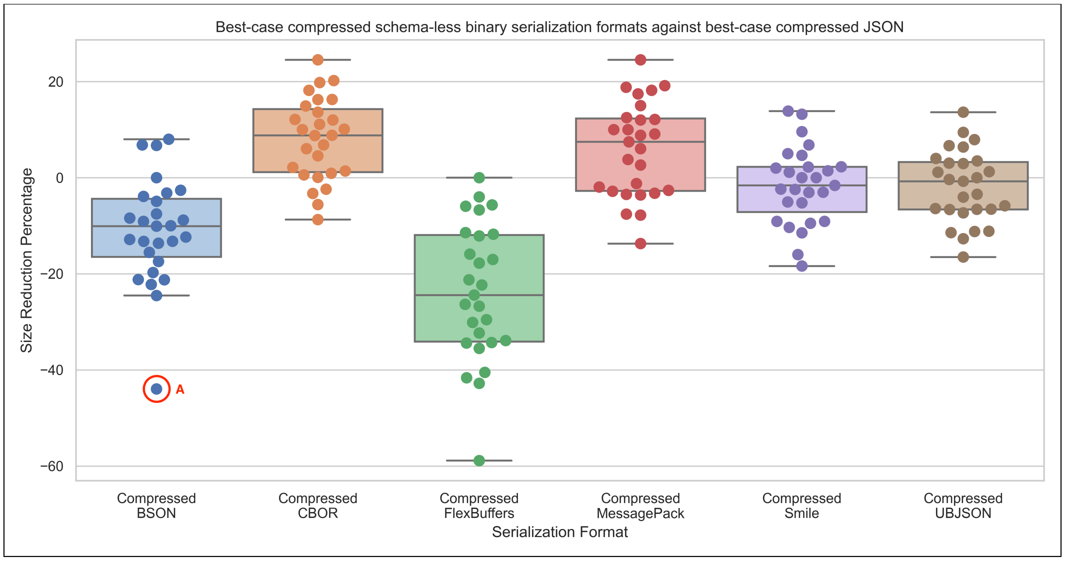

Figure 72: A box plot that demonstrates the size reduction (in percentages) of the selection of schema-less binary serialization specifications listed in Table 7 in their best-case compressed forms given the compression formats listed in Table 9 in comparison to the best-case compressed JSON [17] given the compression formats listed in Table 9 and the input data listed in Table 4 and Table 5.

both compressed and uncompressed schema-driven binary serialization specifications result in positive median and average reduction. Furthermore, compressed sequential schema-driven binary serialization specifications are strictly superior to compressed JSON in all the cases from the input data.

Table 69: A summary of the size reduction results in comparison to the best case scenarios of compressed JSON [17] given the compression formats listed in Table 9 of the selection of schema-diven binary serialization specifications listed in Table 6 against the input data listed in Table 4 and Table 5. See Figure 73 for a visual representation of this data.

Serialization Specification	**Size Reductions in Comparison To Compressed JSON**					**Negative Cases**
	Maximum	**Minimum**	**Range**	**Median**	**Average**	
ASN.1 (PER Unaligned)	98.5%	-222.7%	321.3	75.5%	39%	6 / 27 (22.2%)
Apache Avro (unframed)	100%	-227.3%	327.3	72.7%	39.4%	5 / 27 (18.5%)
Microsoft Bond (Compact Binary v1)	93.2%	-239%	332.1	60.2%	23.4%	6 / 27 (22.2%)
Cap'n Proto	70.1%	-315.6%	385.7	-9.1%	-45.7%	15 / 27 (55.5%)
Cap'n Proto (packed)	86.4%	-267.5%	353.9	50%	17%	8 / 27 (29.6%)
FlatBuffers	54.5%	-486.2%	540.8	-23.4%	-55.4%	17 / 27 (62.9%)
Protocol Buffers	100%	-238.3%	338.3	67%	28.4%	6 / 27 (22.2%)
Apache Thrift (Compact Protocol)	98%	-238.3%	336.3	69.3%	29%	6 / 27 (22.2%)
Averages	**87.6%**	**-279.4%**	**367**	**45.3%**	**9.4%**	**31.9%**

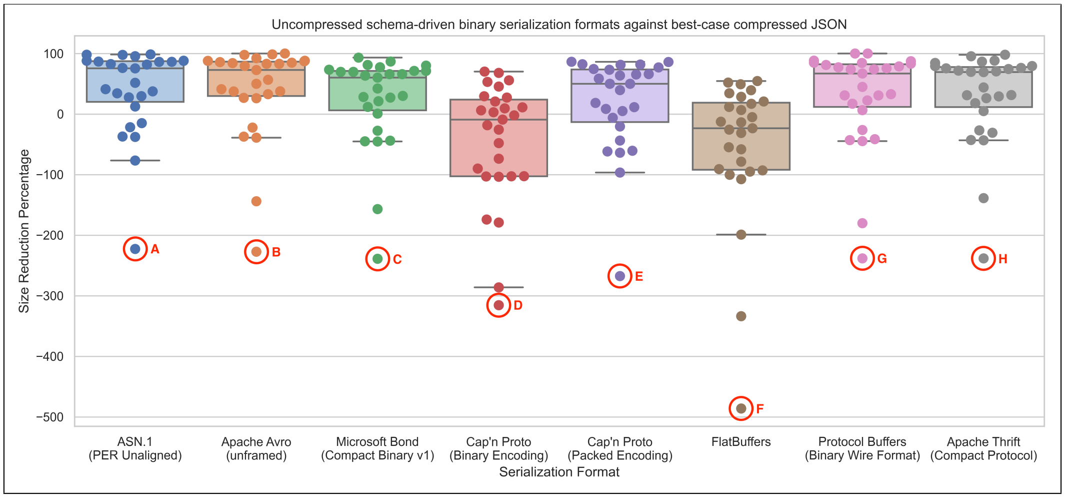

Figure 73: A box plot that demonstrates the size reduction (in percentages) of the selection of schema-driven binary serialization specifications listed in Table 6 in comparison to the best-case compressed JSON [17] given the compression formats listed in Table 9 and the input data listed in Table 4 and Table 5.

Table 70: A summary of the size reduction results of the best case scenarios of compressed schema-driven binary serialization specifications listed in Table 6 in comparison to the best case scenarios of compressed JSON [17] given the compression formats listed in Table 9 and the the input data listed in Table 4 and Table 5. See Figure 74 for a visual representation of this data.

Serialization Specification	**Size Reductions in Comparison To Compressed JSON**					**Negative Cases**
	Maximum	**Minimum**	**Range**	**Median**	**Average**	
Compressed ASN.1 (PER Unaligned)	83.8%	11.2%	72.6	54.5%	51.2%	0 / 27 (0%)
Compressed Apache Avro (unframed)	84%	16.7%	67.3	52.2%	52.3%	0 / 27 (0%)
Compressed Microsoft Bond (Compact Binary v1)	66.3%	8.6%	57.6	42%	40.3%	0 / 27 (0%)
Compressed Cap'n Proto	75.7%	-13.8%	89.4	22.1%	28%	3 / 27 (11.1%)
Compressed Cap'n Proto (packed)	77.2%	-18.1%	95.3	30.2%	32.7%	3 / 27 (11.1%)
Compressed FlatBuffers	60.4%	-68.1%	128.5	10.2%	12.1%	9 / 27 (33.3%)
Compressed Protocol Buffers	80.3%	7.8%	72.5	46.4%	44.6%	0 / 27 (0%)
Compressed Apache Thrift (Compact Protocol)	78.1%	10.3%	67.8	48.9%	46.2%	0 / 27 (0%)
Averages	**75.7%**	**-5.7%**	**81.4**	**38.3%**	**38.4%**	**6.9%**

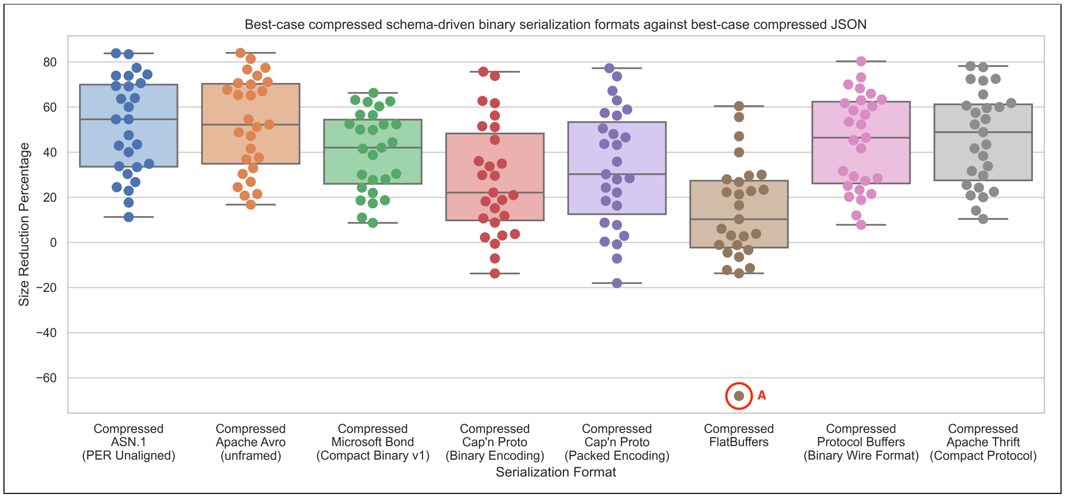

Figure 74: A box plot that demonstrates the size reduction (in percentages) of the selection of schema-driven binary serialization specifications listed in Table 6 in their best-case compressed forms given the compression formats listed in Table 9 in comparison to the best-case compressed JSON [17] given the compression formats listed in Table 9 and the input data listed in Table 4 and Table 5.

8.5 JSON Compatibility

Implementing the benchmark and writing schemas for the set of schema-driven serialization specification revealed that some of the considered schema-driven serialization specifications are not strictly compatible with JSON [17] and required transformations in order for the input data to be accepted by the implementations or the respective schema definition languages when encoding JSON documents from the input data set listed in Table 4 and Table 5. These transformations can be inspected in the benchmark public GitHub repository [124]. These transformations are divided into the following categories:

- **Keys.** The schema definition languages provided by ASN.1 [56], Microsoft Bond [43], Cap'n Proto [69], FlatBuffers [66], Protocol Buffers [31], and Apache Thrift [60] disallow property names that include hyphens, slashes, dollar signs, parenthesis, and periods. Also, ASN.1 [56] disallows property names that start with an underscore and Cap'n Proto [69] disallows property names include underscores and capitalised property names. Furthermore, Protocol Buffers [31] and Apache Thrift [60] disallow property names that equal the reserved keywords *async*, *extends*, *in*, and *with*. To handle these cases, the disallowed properties are renamed to a close variation that the schema language permits.
- **Values.** Protocol Buffers [31] defines the *null* type as an enumeration consisting of a single constant: zero [125]. FlatBuffers [66] does not support a *null* type. When using FlatBuffers [66], we represent this type with an enumeration consisting of a single constant in the same manner as Protocol Buffers [31]. In both cases, we transform any JSON [17] *null* value into zero.
- **Structural.** Neither Microsoft Bond [43], Cap'n Proto [69], FlatBuffers [66], Protocol Buffers [31], nor Apache Thrift [60] support encoding a JSON document that consists of a top level array. In these cases, we move the array into a wrapper structure. FlatBuffers [66] and Protocol Buffers [31] also do not support nested arrays. In these cases, we introduce wrapper structures at every array nesting level. Finally, ASN.1 [56], Microsoft Bond [43], Cap'n Proto [69], FlatBuffers [66], Protocol Buffers [31], and Apache Thrift [60] do not support heterogenous arrays of non-composite types. In these cases, we convert the heterogenous arrays into arrays of union structures. Microsoft Bond [43] does not support union types and in this case we introduce a structure consisting of optional fields.

[124] `https://github.com/jviotti/binary-json-size-benchmark`

[125] `https://github.com/protocolbuffers/protobuf/blob/master/src/google/protobuf/struct.proto`

Additionally, the use of unions in FlatBuffers [66] requires the introduction of an additional textual property to signify the union choice. In order not to put this specification at a disadvantage, we encode the fixed-length heterogenous array as tables where their property names correspond to the array indexes.

The type of transformations that were necessary for each JSON document from the input data defined in Table 4 and Table 5 are listed in Table 71. In summary, every schema-less binary serialization specifications listed in Table 7 is compatible with the input data set. In terms of schema-driven specifications, only Apache Avro [27] is strictly compatible with the input data set.

Table 71: A summary of the transformations needed to serialize the input data JSON documents listed in Table 4 and Table 5 using the set of binary serialization specifications listed in Table 6 and Table 7. The JSON documents from Table 4 and Table 5 that are not present in this table did not require any type of transformation. Each letter signifies the type of required transformation as defined in this section. The letter K stands for *Keys*, the letter V stands for *Values*, and the letter S stands for *Structural*.

Input Data	**ASN.1**	**Apache Avro**	**Microsoft Bond**	**BSON**	**Cap'n Proto**	**CBOR**	**FlatBuffers**	**FlexBuffers**	**MessagePack**	**Protocol Buffers**	**Smile**	**Apache Thrift**	**UBJSON**
Tier 1 NRF	K		K		K		K			K		K	
Tier 1 NRN	K		K		K		K			K		K	
Tier 1 TRF	K		K		K		K			K		K	
Tier 1 TRN	K+S		K+S		K+S		K+S			K+S		K+S	
Tier 1 TNF										K		K	
Tier 1 BRF							V			V			
Tier 1 BRN	K		K		K		K			K		K	
Tier 1 BNN	K		K		K		K			K		K	
Tier 2 NRN							S			S			
Tier 2 NNF					K								
Tier 2 NNN			S		K+S		S			S		S	
Tier 2 TNF					K								
Tier 2 TNN	K		K		K		K			K		K	
Tier 2 BRF					K		V			V			
Tier 3 NRF	K+S		K+S		K+S		K+S			K+S		K+S	
Tier 3 TRF	K+S		K+S		K+S		K+S			K+S		K+S	
Tier 3 TRN	K		K		K		K			K		K	
Tier 3 BRF					K		V			V			
Tier 3 TNF	K		K		K		K			K		K	

9 Future Work

In this paper, we present the results of a comprehensive benchmark of 13 JSON-compatible schema-driven and schema-less binary serialization specifications across 27 real-world JSON documents test cases across industries.

Our findings provide a number of conclusions. When we investigated how JSON-compatible schema-less binary serialization specifications compare to JSON in terms of space-efficiency, we found that using MessagePack [29] on *Tier 1 Minified* < 100 *bytes* and *Tier 2 Minified* $\geq 100 < 1000$ *bytes* JSON documents, Smile [55] on *Tier 3 Minified* ≥ 1000 *bytes* JSON documents, and FlexBuffers [67] on JSON documents with high-redundancy of *textual* values increases space-efficiency. When we investigated how JSON-compatible schema-driven binary serialization specifications compare to JSON and JSON-compatible schema-less binary serialization specifications in terms of space-efficiency, we found that ASN.1 PER Unaligned [57] and Apache Avro (unpacked) [27] are space-efficient in comparison to schema-less binary serialization specifications in almost all cases. When we investigated how JSON-compatible sequential binary serialization specifications to compare to JSON-compatible pointer-based binary serialization specifications in terms of space-efficiency, we found that sequential binary serialization specifications are typically more space-efficient than pointer-based binary serialization specifications, independent of whether they are schema-less or schema-driven. When we investigated how compressed JSON compares to uncompressed and compressed JSON-compatible binary serialization specifications, we found that in comparison to compressed JSON, both compressed and uncompressed schema-less binary serialization specifications result in negative median and average size reductions. However, both compressed and uncompressed schema-driven binary serialization specifications result in positive median and average reduction. Furthermore, compressed sequential schema-driven binary serialization specifications are strictly superior to compressed JSON in all the cases from the input data.

Based on our findings, we believe there is room to augment the input data set to include JSON documents that match the 9 missing taxonomy categories described in subsection 5.1 and to increase the sample proportionality. We hope to encourage contributions to our open-source space-efficiency benchmark automation software for general improvements and support for new JSON-compatible binary serialization specifications. Using our learnings, we hope to propose a new JSON-compatible binary serialization specification with better space-efficiency characteristics.

Acknowledgments and Disclosure of Funding

Thanks to OSS Nokalva [126] for offering access to and help for using their proprietary ASN-1Step ASN.1 [56] implementation.

References

[1] Berkman Center for Internet & Society at Harvard Law School 2003. *RSS 2.0 Specification*. Berkman Center for Internet & Society at Harvard Law School. `https://www.rssboard.org/rss-2-0-1`

[2] Mohamed-Amine Baazizi, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. 2019. Parametric schema inference for massive JSON datasets. *The VLDB Journal* 28, 4 (2019), 497–521.

[3] Matěj Bartík, Sven Ubik, and Pavel Kubalik. 2015. LZ4 compression algorithm on FPGA. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*. 179–182. `https://doi.org/10.1109/ICECS.2015.7440278`

[4] Amrit Kumar Biswal and Obada Almallah. 2019. *Analytical Assessment of Binary Data Serialization Techniques in IoT Context*. Master's thesis. Dipartimento di Elettronica, Informazione e Bioingegneria.

[5] C. Bormann and P. Hoffman. 2013. *Concise Binary Object Representation (CBOR)*. RFC. IETF. `https://doi.org/10.17487/RFC7049`

[6] Pierre Bourhis, Juan L. Reutter, Fernando Suárez, and Domagoj Vrgoč. 2017. JSON: Data Model, Query Languages and Schema Specification. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Chicago, Illinois, USA) *(PODS '17)*. Association for Computing Machinery, New York, NY, USA, 123–135. `https://doi.org/10.1145/3034786.3056120`

[7] Steven Braeger. 2016. *Universal Binary JSON Specification*. UBJSON. `https://ubjson.org`

[8] T. Bray. 2014. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC. IETF. `https://doi.org/10.17487/RFC8259`

[9] P. Bryan. 2013. *JavaScript Object Notation (JSON) Patch*. RFC. IETF. `https://doi.org/10.17487/RFC6902`

[10] P. Bryan. 2013. *JavaScript Object Notation (JSON) Pointer*. RFC. `https://doi.org/10.17487/RFC6901`

[11] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, and T. Schaub. 2016. *The GeoJSON Format*. RFC. IETF. `https://doi.org/10.17487/RFC7946`

[12] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2013. Discovering Implicit Schemas in JSON Data. In *Web Engineering*, Florian Daniel, Peter Dolog, and Qing Li (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 68–83.

[13] Yann Collet and Murray Kucherawy. 2021. *Zstandard Compression and the 'application/zstd' Media Type*. RFC. IETF. `https://doi.org/10.17487/RFC8878`

[14] The Unicode Consortium. 2019. *The Unicode Standard, Version 12.1.0*. Standard. The Unicode Consortium, Mountain View, CA.

[15] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2016. JSONDiscoverer: Visualizing the schema lurking behind JSON documents. *Knowledge-Based Systems* 103 (2016), 52–55. `https://doi.org/10.1016/j.knosys.2016.03.020`

[16] P. Deutsch. 1996. *GZIP file format specification version 4.3*. RFC. `https://doi.org/10.17487/RFC1952`

[126] `https://www.ossnokalva.com`

[17] ECMA. 2017. *ECMA-404: The JSON Data Interchange Syntax*. ECMA, Geneva, CH. https://www.ecma-international.org/publications/standards/Ecma-404.htm

[18] ECMA. 2021. *ECMA-262: ECMAScript 2021 language specification*. ECMA, Geneva, CH. https://www.ecma-international.org/publications/standards/Ecma-262.htm

[19] Hamza Ed-douibi, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2017. Example-Driven Web API Specification Discovery. In *Modelling Foundations and Applications*, Anthony Anjorin and Huáscar Espinoza (Eds.). Springer International Publishing, Cham, 267–284.

[20] G. Court F. Galiegue, K. Zyp. [n.d.]. *JSON Schema: core definitions and terminology*. IETF. https://tools.ietf.org/html/draft-zyp-json-schema-04

[21] Steve Faulkner, Arron Eicholz, Travis Leithead, Alex Danilo, and Sangwhan Moon. 2021. *HTML 5.2*. W3C Recommendation. W3C. https://www.w3.org/TR/html52/.

[22] Jianhua Feng and Jinhong Li. 2013. Google protocol buffers research and application in online game. In *IEEE Conference Anthology*. IEEE, China, 4.

[23] Paolo Ferragina and Giovanni Manzini. 2010. On Compressing the Textual Web. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining* (New York, New York, USA) *(WSDM '10)*. Association for Computing Machinery, New York, NY, USA, 391–400. https://doi.org/10.1145/1718487.1718536

[24] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. 1999. *Hypertext Transfer Protocol – HTTP/1.1*. RFC. https://doi.org/10.17487/RFC2616

[25] R. Fielding and J. Reschke. 2014. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC. https://doi.org/10.17487/RFC7231

[26] Working Group for Floating-Point Arithmetic. 2019. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)* 1, 1 (2019), 1–84.

[27] The Apache Software Foundation. 2012. *Apache Avro™ 1.10.0 Specification*. The Apache Software Foundation. https://avro.apache.org/docs/current/spec.html

[28] A. A. Frozza, R. d. S. Mello, and F. d. S. d. Costa. 2018. An Approach for Schema Extraction of JSON and Extended JSON Document Collections. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. IEEE, Salt Lake City, Utah, 356–363.

[29] Sadayuki Furuhashi, Satoshi Tagomori, Yuichi Tanikawa, Stephen Colebourne, Stefan Friesel, René Kijewski, Michael Cooper, Uenishi Kota, and Gabe Appleton. 2013. *MessagePack Specification*. MessagePack. https://github.com/msgpack/msgpack/blob/master/spec.md

[30] Bruno Gil and Paulo Trezentos. 2011. Impacts of Data Interchange Formats on Energy Consumption and Performance in Smartphones. In *Proceedings of the 2011 Workshop on Open Source and Design of Communication* (Lisboa, Portugal) *(OSDOC '11)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/2016716.2016718

[31] Google. 2020. *Protocol Buffers Version 3 Language Specification*. Google. https://developers.google.com/protocol-buffers/docs/reference/proto3-spec

[32] Jean Carlo Hamerski, Anderson RP Domingues, Fernando G Moraes, and Alexandre Amory. 2018. Evaluating serialization for a publish-subscribe based middleware for MPSoCs. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, IEEE, Bordeaux, 773–776.

[33] Sam Harrison, Abhishek Dasgupta, Simon Waldman, Alex Henderson, and Christopher Lovell. 2021. How reproducible should research software be? https://doi.org/10.5281/zenodo.4761867

[34] David A Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40, 9 (1952), 1098–1101.

[35] their environments ISO/IEC JTC 1/SC 22 Programming languages and system software interfaces. 2002. *Z formal specification notation — Syntax, type system and semantics*. Standard. International Organization for Standardization.

[36] K. Jahed and J. Dingel. 2019. Enabling Model-Driven Software Development Tools for the Internet of Things. In *2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE)*. IEEE, Montreal, QC, Canada, 93–99. `https://doi.org/10.1109/MiSE.2019.00022`

[37] Uthayakumar Jayasankar, Vengattaraman Thirumal, and Dhavachelvan Ponnurangam. 2021. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *Journal of King Saud University - Computer and Information Sciences* 33, 2 (2021), 119–140. `https://doi.org/10.1016/j.jksuci.2018.05.006`

[38] Meike Klettke, Uta Störl, and Stefanie Scherzinger. 2015. Schema extraction and structural outlier detection for JSON-based nosql data stores. In *Datenbanksysteme für Business, Technologie und Web (BTW 2015)*, Thomas Seidl, Norbert Ritter, Harald Schöning, Kai-Uwe Sattler, Theo Härder, Steffen Friedrich, and Wolfram Wingerath (Eds.). Gesellschaft für Informatik e.V., Bonn, 425–444.

[39] Ronny Krashinsky. 2003. Efficient web browsing for mobile clients using HTTP compression. *See: http://www. cag. lcs. mit. edu/˜ ronny/classes/httpcomp. pdf* (2003).

[40] Pavel Kyurkchiev. 2015. Integrating a System for Symbol Programming of Real Processes with a Cloud Service. , 8 pages.

[41] K Maeda. 2012. Performance evaluation of object serialization libraries in XML, JSON and binary formats. , 177–182 pages.

[42] Dwight Merriman, Stephen Steneker, Hannes Magnusson, Luke Lovett, Kevin Albertson, Kay Kim, and Allison Reinheimer Moore. 2020. *BSON Specification Version 1.1*. MongoDB. `http://bsonspec.org/spec.html`

[43] Microsoft. 2018. *Bond Compiler 0.12.0.1*. Microsoft. `https://microsoft.github.io/bond/manual/compiler.html`

[44] M. Nottingham and R. Sayre. 2005. *The Atom Syndication Format*. RFC. IETF. `https://doi.org/10.17487/RFC4287`

[45] PM Parekar and SS Thakare. 2014. Lossless data compression algorithm–a review. *International Journal of Computer Science & Information Technologies* 5, 1 (2014).

[46] Roger D Peng. 2011. Reproducible research in computational science. *Science* 334, 6060 (2011), 1226–1227.

[47] Bo Petersen, Henrik Bindner, Shi You, and Bjarne Poulsen. 2017. Smart grid serialization comparison: Comparision of serialization for distributed control in the context of the internet of things. In *2017 Computing Conference*. IEEE, IEEE, London, UK, 1339–1346.

[48] S. Popić, D. Pezer, B. Mrazovac, and N. Teslić. 2016. Performance evaluation of using Protocol Buffers in the Internet of Things communication. In *2016 International Conference on Smart Systems and Technologies (SST)*. International Conference on Smart Systems and Technologies, Osijek, HR, 261–265.

[49] M. A. Pradana, A. Rakhmatsyah, and A. A. Wardana. 2019. Flatbuffers Implementation on MQTT Publish/Subscribe Communication as Data Delivery Format. In *2019 6th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*. IEEE, Bandung, Indonesia, 142–146. `https://doi.org/10.23919/EECSI48112.2019.8977050`

[50] D. P. Proos and N. Carlsson. 2020. Performance Comparison of Messaging Protocols and Serialization Formats for Digital Twins in IoV. In *2020 IFIP Networking Conference (Networking)*. IEEE, Paris, France, 10–18.

[51] Hilary Putnam. 1957. Three-valued logic. *Philosophical Studies* 8, 5 (1957), 73–80.

[52] David Ross and Mike West. 2016. *Entry Point Regulation*. W3C Working Group Note. W3C. https://w3c.github.io/webappsec-epr/.

[53] Kristina Sahlmann, Alexander Lindemann, and Bettina Schnor. 2018. Binary Representation of Device Descriptions: CBOR versus RDF HDT. *Technische Universität Braunschw: Braunschweig, Germany* 0, 0 (2018), 4.

[54] Y. Sakamoto, S. Matsumoto, S. Tokunaga, S. Saiki, and M. Nakamura. 2015. Empirical study on effects of script minification and HTTP compression for traffic reduction. In *2015 Third International Conference on Digital Information, Networking, and Wireless Communications (DINWC)*. 127–132. `https://doi.org/10.1109/DINWC.2015.7054230`

[55] Tatu Saloranta. 2017. *Efficient JSON-compatible binary format: "Smile"*. FasterXML. `https://github.com/FasterXML/smile-format-specification/blob/master/smile-specification.md`

[56] ITU Telecommunication Standardization Sector. 2015. *Abstract Syntax Notation One (ASN.1): Specification of basic notation*. Standard. ITU Telecommunication Standardization Sector.

[57] ITU Telecommunication Standardization Sector. 2015. *ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*. Standard. ITU Telecommunication Standardization Sector.

[58] Edleno Silva de Moura, Gonzalo Navarro, Nivio Ziviani, and Ricardo Baeza-Yates. 2000. Fast and Flexible Word Searching on Compressed Text. *ACM Trans. Inf. Syst.* 18, 2 (April 2000), 113–139. `https://doi.org/10.1145/348751.348754`

[59] Brent Simmons and Manton Reece. 2017. *JSON Feed Version 1*. `https://jsonfeed.org/version/1`

[60] Mark Slee, Aditya Agarwal, and Marc Kwiatkowski. 2007. Thrift: Scalable cross-language services implementation. *Facebook White Paper* 5 (2007), 8.

[61] Simon Stewart and David Burns. 2020. *WebDriver*. W3C Working Draft. W3C.

[62] James A. Storer and Thomas G. Szymanski. 1982. Data Compression via Textual Substitution. *J. ACM* 29, 4 (Oct. 1982), 928–951. `https://doi.org/10.1145/322344.322346`

[63] Audie Sumaray and S Makki. 2012. A comparison of data serialization formats for optimal efficiency on a mobile platform. , 6 pages.

[64] Pavel Conto˘s˘ and Martin Svoboda. 2020. JSON Schema Inference Approaches. In *Advances in Conceptual Modeling: ER 2020 Workshops CMAI, CMLS, CMOMM4FAIR, CoMoNoS, EmpER, Vienna, Austria, November 3–6, 2020, Proceedings*. Springer Nature, Springer, Vienna, Austria, 173.

[65] Jordi van Liempt. 2020. *CityJSON: does (file) size matter?* Master's thesis. Department of Urbanism, Faculty of the Built Environment & Architecture.

[66] Wouter van Oortmerssen. 2014. *FlatBuffers: Writing a Schema*. Google. `https://google.github.io/flatbuffers/flatbuffers_guide_writing_schema.html`

[67] Wouter van Oortmerssen. 2017. *FlexBuffers*. Google. `https://google.github.io/flatbuffers/flexbuffers.html`

[68] J. Vanura and P. Kriz. 2018. Perfomance evaluation of Java, JavaScript and PHP serialization libraries for XML, JSON and binary formats. , 166–175 pages.

[69] Kenton Varda. 2013. *Cap'n Proto Schema Language*. Sandstorm. `https://capnproto.org/language.html`

[70] Juan Cruz Viotti. 2022. *jviotti/binary-json-size-benchmark:*. `https://doi.org/10.5281/zenodo.5829569`

[71] Juan Cruz Viotti and Mital Kinderkhedia. 2022. A Survey of JSON-compatible Binary Serialization Specifications. arXiv:2201.02089 [cs.DB]

[72] Canggih Wibowo. 2011. Evaluation of Protocol Buffers as Data Serialization Format for Microblogging Communication.

[73] Martin Wischenbart, Stefan Mitsch, Elisabeth Kapsammer, Angelika Kusel, Birgit Pröll, Werner Retschitzegger, Wieland Schwinger, Johannes Schönböck, Manuel Wimmer, and Stephan Lechner. 2012. User Profile Integration Made Easy: Model-Driven Extraction and Transformation of Social Network Schemas. In *Proceedings of the 21st International Conference on World Wide Web* (Lyon, France) *(WWW '12 Companion)*. Association for Computing Machinery, New York, NY, USA, 939–948. `https://doi.org/10.1145/2187980.2188227`

[74] A. Wright, H. Andrews, and B. Hutton. 2019. *JSON Schema: A Media Type for Describing JSON Documents*. Technical Report. IETF. `https://tools.ietf.org/html/draft-handrews-json-schema-02`

[75] A. Wright, H. Andrews, and B. Hutton. 2020. *JSON Schema: A Media Type for Describing JSON Documents*. Technical Report. IETF. `https://tools.ietf.org/html/draft-bhutton-json-schema-00`

[76] J. Ziv and A. Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* 23, 3 (1977), 337–343. `https://doi.org/10.1109/TIT.1977.1055714`