Benchmarking JSON BinPack

Juan Cruz Viotti*  
Department of Computer Science  
University of Oxford  
Oxford, GB OX1 3QD  
jv@jviotti.com

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 benchmark results for a pre-production implementation of a novel serialization specification: JSON BinPack. JSON BinPack is a schema-driven and schema-less sequential binary serialization specification based on JSON Schema. It is rich in diverse encodings, and is developed to improve network performance and reduce the operational costs of Internet-based software systems. We present benchmark results for 27 JSON documents and for each plot, we show the schema-driven and schema-less serialization specifications that produce the smallest bit-strings. Through extensive plots and statistical comparisons, we show that JSON BinPack in schema-driven mode is as space-efficient or more space-efficient than every other serialization specification for the 27 documents under consideration. In comparison to JSON, JSON BinPack in schema-driven mode provides a median and average size reductions of 86.7% and 78.7%, respectively. We also show that the schema-less mode of the JSON BinPack binary serialization specification is as space-efficient or more space-efficient than every other schema-less serialization specification for the 27 documents under consideration. In comparison to JSON, JSON BinPack in schema-less mode provides a median and average size reductions of 30.6% and 30.5%, respectively. Unlike other considered schema-driven binary serialization specifications, JSON BinPack in schema-driven mode is space-efficient in comparison to best-case compressed JSON in terms of the median and average with size reductions of 76.1% and 66.8%, respectively. We have made our benchmark results available at jviotti/binary-json-size-benchmark on GitHub.

1 Introduction

For consumers of Internet-based software systems, substandard network performance results in impaired user experience given that an increasing amount of software is now accessed over the Internet. This type of software systems are particularly sensitive to substandard network performance. Additionally, software systems that operate over the Internet typically rely on infrastructure that charges for inbound or outbound network communication. Given the decentralized architecture and complex dynamics of the Internet, network communications are unpredictable and often unreliable. Therefore, transmitting data over the network directly translates to operational expenses.

To facilitate interoperability, Internet-based software systems transmit information using data serialization specifications such as JSON [6] and XML [18]. [33] identifies JSON [6] as the dominant data interchange standard in the context of cloud software systems. However, it concludes that JSON is neither a space-efficient nor a runtime-efficient serialization specification. [14] argues that the bottleneck of Internet network communication is the time of transmission, making the runtime-efficiency

*https://www.jviotti.com

Preprint. Under review.
aspect of the choice of a serialization specification irrelevant in such context. Therefore, the choice
of a serialization specification and its space-efficiency characteristics are crucial for improving net-
work performance and reducing the operational costs of Internet-based software systems.

In the pursuit of a solution to enhance network performance and reduce operational costs, our previ-
ous works [33] [31] explore in depth a set of 13 schema-driven and schema-less JSON-compatible
binary serialization specifications with different space-efficiency, runtime-efficiency and architec-
tural characteristics: ASN.1 [23], Apache Avro [9], Microsoft Bond [16], Cap’n Proto [29], Flat-
Buffers [27], Protocol Buffers [12], and Apache Thrift [25], BSON [15], CBOR [3], FlexBuffers
[28], MessagePack [10], Smile [21] and UBJSON [4], and proposes JSON BinPack - a novel open-
source binary serialization specification that has a strong focus on space-efficiency, is strictly JSON-
compatible, is a hybrid between schema-driven and schema-less and relies on the industry-standard
schema language for describing JSON documents: JSON Schema [34] (upcoming paper). Using
the same set of JSON-compatible binary serialization specifications and taking data compression
into consideration, we produced a comprehensive space-efficiency benchmark [31] that is based on
a representative set of 27 real-world JSON documents and analyzed the results across different lens
to understand in which cases and why certain serialization specifications outperform others.

1.1 Paper Organization

This paper is organized as follows. In section 1, we discuss related literature in the context of
space-efficient binary serialization specifications and space-efficiency benchmarking. We propose a
set of space-efficiency-oriented research questions for the novel JSON BinPack binary serialization
specification and we state our contributions. In section 2, we review the benchmarking methodology
introduced in [31] and apply it to evaluate the JSON BinPack binary serialization specification. We
then describe the alternative serialization specifications we compare JSON BinPack to, linking them
to the schema definitions used as part of the benchmark study and expand on our strategy to provide
a fair benchmark. In section 3, we present the results of extending the benchmark study introduced
in [31] to include the JSON BinPack binary serialization specification and state how our experiments
could be reproduced. In section 4, we critically evaluate the benchmark results in terms of each of
the research questions proposed in subsection 1.3. In section 5, we conclude with an assessment of
future work to be done for the JSON BinPack binary serialization specification.

1.2 Related Literature

In [33], we introduce the problem of data serialization, summarize the state-of-the-art in JSON-
compatible data serialization literature, discuss the history of JSON [6], its relevance, characteristics
and shortcomings. Next, we study the history, the advantages, the characteristics and encodings of 13
popular 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 and discuss the problem of schema
evolution in the context of JSON-compatible binary schema-driven serialization specifications.

As a continuation of this work, [31] makes a case for the importance of space-efficiency in the con-
text of data transmission as an approach to improve network performance and reduce operational
costs. To evaluate the space-efficiency characteristics of available JSON-compatible serialization
specifications, [31] presents a comprehensive and representative benchmark of JSON-compatible
binary serialization specifications. The benchmark involves the 13 schema-driven and schema-
less binary serialization specifications studied in [33]. The input data consists of a set of 27 real-
world JSON documents obtained from the SchemaStore open-source dataset. The pre-production
benchmark software is open-source and is designed to be continuously extended with new JSON-
compatible serialization specifications and input JSON documents.

[31] concludes that ASN.1 PER Unaligned [23] and Apache Avro [9] are space-efficient in compa-
rison to JSON and other schema-driven and schema-less JSON-compatible binary serialization
specifications in most cases. Furthermore, space-efficient schema-driven sequential binary serial-
ization specifications tend to outperform JSON general purpose data compression, especially on
small JSON documents. However, no considered binary serialization specification is strictly supe-
rior than JSON nor the other considered serialization specifications in every case. Out of the selec-
tion of schema-driven binary serialization specifications, Apache Avro [9] is the only serialization specification that was found to be strictly JSON-compatible.

To methodically solve the input data selection process and provide a fair representative benchmark using the smallest possible set of input data, [32] introduces a formal tiered taxonomy for JSON documents. This taxonomy consists of 36 categories classified as Tier 1, Tier 2 and Tier 3 as a common basis to class JSON documents based on characteristics that are relevant in the context of data serialization such as size, type of content, structural and redundancy criteria. The categories of the taxonomy are visually represented in Figure 1.

![Figure 1: [31] introduces a formal taxonomy to class JSON documents that consists of 36 categories.](image)

1.3 Research Questions

This paper aims to evaluate the space-efficiency characteristics of the JSON BinPack pre-production implementation against the JSON-compatible schema-less and schema-driven binary serialization specifications and general purpose data compressors studied in [33] and [31].

This benchmark aims to answer the following set of research questions:

- **Q1**: How does JSON BinPack in schema-driven mode compare to JSON in terms of space-efficiency?
• **Q2**: How does JSON BinPack in schema-less mode compare to JSON in terms of space-efficiency?

• **Q3**: How does JSON BinPack in schema-driven and schema-less mode compare to compressed JSON?

### 1.4 Contributions

We believe that the space-efficiency benchmark introduced in [31] is the first of its kind to produce a comprehensive, reproducible, extensible and open-source study that considers a large and representative input dataset of real-world JSON documents across industries and takes data compression into consideration.

Our main contribution in this paper is the extension of the benchmark software developed in [31] to compare the novel JSON BinPack binary serialization specification against the popular JSON-compatible serialization specifications studied in [33].

### 2 Methodology

To evaluate the space-efficiency characteristics of JSON BinPack in comparison to the JSON-compatible binary serialization specifications and general-purpose data compressors studied in [31] and [33], we benchmark the JSON BinPack pre-production implementation using the open-source space-efficiency framework written in [31]. Our approach is the following:

1. **JSON BinPack.** We extended the open-source automated benchmark software implemented in [31] to recognize JSON BinPack as a JSON-compatible binary serialization specification.

2. **Schema Definitions.** We wrote both *strict* and *loose* JSON Schema [35] definitions for the 27 input JSON [6] documents selected in [31]. In total, we wrote 54 schema definitions.

3. **Benchmarking JSON BinPack against Alternatives.** We used the automated benchmark software to serialize and de-serialize the 27 real-world JSON [6] documents selected in [31] with JSON BinPack. [33] proposes the idea of schema-less as a subset of schema-driven. Under this concept, we considered serializing the input data using a loose schema definition that matches any instance, thus executing the JSON BinPack binary specification in **schema-less** mode. Conversely, we considered serializing the input data using the strict schema definition, thus executing the JSON BinPack binary specification in **schema-driven** mode.

4. **Analyzing the Results.** We analyzed the benchmark results of JSON BinPack in comparison to the 14 JSON-compatible serialization specifications and encodings listed in Table 1 and Table 2 by visualizing and comparing them.

5. **Conclusions.** We made deductions based on the benchmark results analyzing how JSON BinPack compared to the 14 JSON-compatible serialization specifications and encodings listed in Table 1 and Table 2 in terms of space-efficiency and JSON-compatibility.

### 2.1 Serialization Specifications

The following benchmark as referred to in [31] compares **9 schema-driven** JSON-compatible binary serialization specifications and encodings listed in Table 1 and **7 schema-less** JSON-compatible binary serialization specifications and encodings listed in Table 2, with the exception of JSON BinPack.

---

2https://avro.apache.org/docs/current/spec.html#binary_encoding
3https://microsoft.github.io/bond/reference/cpp/compact__binary_8h_source.html
4https://capnproto.org/encoding.html#packing
5https://capnproto.org/encoding.html
6https://google.github.io/flatbuffers/flatbuffers_internals.html
7https://developers.google.com/protocol-buffers/docs/encoding
8https://github.com/apache/thrift/blob/master/doc/specs/thrift-compact-protocol.md
Table 1: The selection of schema-driven JSON-compatible binary serialization specifications based on our previous work [31].

| Specification | Implementation | Encoding | License     |
|---------------|----------------|----------|-------------|
| ASN.1         | OSS ASN-1Step Version 10.0.2 | PER Unaligned [23] | Proprietary |
| Apache Avro   | Python avro (pip) 1.10.0 | Binary Encoding \textsuperscript{2} with no framing | Apache-2.0   |
| Microsoft Bond| C++ library 9.0.4 | Compact Binary v1 \textsuperscript{3} | MIT         |
| Cap’ n Proto  | capnp command-line tool 0.8.0 | Binary Encoding \textsuperscript{4} | MIT         |
| Cap’ n Proto  | capnp command-line tool 0.8.0 | Packed Encoding \textsuperscript{5} | MIT         |
| FlatBuffers   | flatc command-line tool 1.12.0 | Binary Wire Format \textsuperscript{6} | Apache-2.0   |
| JSON BinPack  | Pre-production implementation v1.1.2 | Binary Encoding | Apache-2.0   |
| Protocol Buffers | Python protobuf (pip) 3.15.3 | Binary Wire Format \textsuperscript{7} | 3-Clause BSD |
| Apache Thrift | Python thrift (pip) 0.13.0 | Compact Protocol \textsuperscript{8} | Apache-2.0   |

Table 2: The selection of schema-less JSON-compatible binary serialization specifications based on our previous work [31].

| 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   |
| JSON BinPack  | Pre-production implementation v1.1.2 | 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   |

2.2 Schema Definitions

For brevity, we do not include the schema definitions for each input JSON document listed in [31]. The schema definitions can be found on the GitHub repository implemented as part of the benchmark study \textsuperscript{9}.

2.3 Fair Benchmarking

We aim to provide a fair benchmark. Thus the resulting bit-strings are ensured to be lossless encodings of the respective input JSON documents.

The implemented benchmark program validates that for each combination of serialization specification listed in subsection 2.1 and input JSON document listed in [31], 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.

\textsuperscript{9}https://github.com/jviotti/binary-json-size-benchmark
3 Benchmark

In this section, we present the benchmark results for the 27 JSON [6] documents considered by [31]. For each document, we present the plots and the related analysis. The schema-driven and schema-less serialization specifications that produce the smallest bit-strings for the corresponding documents are shown in full opacity. JSON BinPack (Schema-driven) corresponds to executing JSON BinPack with a strict schema definition and JSON BinPack (Schema-less) corresponds to executing JSON BinPack with the wildcard schema definition that matches any instance.

| Description                                               | Section       | Category |
|-----------------------------------------------------------|---------------|----------|
| JSON-e templating engine sort example                     | subsection 3.1| TNRF     |
| JSON-e templating engine reverse sort example             | subsection 3.2| TNRN     |
| CircleCI definition (blank)                               | subsection 3.3| TNNF     |
| CircleCI matrix definition                                | subsection 3.4| TNNN     |
| Grunt.js “clean” task definition                          | subsection 3.5| TTRF     |
| CommitLint configuration                                  | subsection 3.6| TTRN     |
| TSLint linter definition (extends only)                   | subsection 3.7| TTFN     |
| ImageOptimizer Azure Webjob configuration                 | subsection 3.8| TTNN     |
| SAP Cloud SDK Continuous Delivery Toolkit configuration   | subsection 3.9| TBRF     |
| TSLint linter definition (multi-rule)                     | subsection 3.10| TBRN    |
| CommitLint configuration (basic)                          | subsection 3.11| TBNF    |
| TSLint linter definition (basic)                          | subsection 3.12| TBNR    |
| GeoJSON example JSON document                             | subsection 3.13| SNRN    |
| OpenWeatherMap API example JSON document                  | subsection 3.14| SNNF    |
| OpenWeather Road Risk API example                         | subsection 3.15| SNNN    |
| TravisCI notifications configuration                      | subsection 3.16| STRF    |
| Entry Point Regulation manifest                           | subsection 3.17| STRN    |
| JSON Feed example document                                | subsection 3.18| STNF    |
| GitHub Workflow Definition                                | subsection 3.19| STNN    |
| GitHub FUNDING sponsorship definition (empty)             | subsection 3.20| SBRF    |
| ECMAScript module loader definition                       | subsection 3.21| SBNF    |
| ESLint configuration document                              | subsection 3.22| LNRF    |
| NPM Package.json Linter configuration manifest             | subsection 3.23| LTRF    |
| .NET Core project.json                                    | subsection 3.24| LTRN    |
| NPM Package.json example manifest                          | subsection 3.25| LTNF    |
| JSON Resume                                               | subsection 3.26| LTNN    |
| Nightwatch.js Test Framework Configuration                | subsection 3.27| LBRF    |
3.1 JSON-e Templating Engine Sort Example

JSON-e\textsuperscript{10} is an open-source JSON-based templating engine created by Mozilla as part of the TaskCluster\textsuperscript{11} project, the open-source task execution framework that supports Mozilla's continuous integration and release processes. In Figure 2, we demonstrate a Tier 1 minified < 100 bytes numeric redundant flat (Tier 1 NRF from [31]) JSON document that consists of an example JSON-e template definition to sort an array of numbers.

![Figure 2: The space-efficiency benchmark results for the JSON-e Templating Engine Sort Example test case selected from the SchemaStore open-source dataset test suite in [31].](image)

The smallest bit-string produced by JSON BinPack (Schema-driven) results in a 11.1\% (8 bytes) size reduction compared to the next best performing specification: Apache Avro [9] (9 bytes). JSON BinPack (Schema-driven) achieves a 61.9\% size reduction compared to the best performing schema-less serialization specifications: JSON BinPack (Schema-less), CBOR [3] and MessagePack [10]. Additionally, JSON BinPack (Schema-less) (21 bytes), along with CBOR [3] and MessagePack [10], produce the smallest bit-string for schema-less binary serialization specifications, resulting in a 22.2\% size reduction compared to the next best performing specification: Smile [21] (27 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (34 bytes), JSON BinPack (Schema-driven) (8 bytes) and JSON BinPack (Schema-less) (21 bytes) achieve a 76.4\% and 38.2\% size reduction, respectively. In comparison to best-case compressed JSON [6] (53 bytes), JSON BinPack (Schema-driven) (8 bytes) and JSON BinPack (Schema-less) (21 bytes) achieve a 84.9\% and 60.3\% size reduction, respectively.

\textsuperscript{10}\url{https://github.com/taskcluster/json-e}

\textsuperscript{11}\url{https://taskcluster.net}
3.2 JSON-e Templating Engine Reverse Sort Example

JSON-e is an open-source JSON-based templating engine created by Mozilla as part of the TaskCluster project, the open-source task execution framework that supports Mozilla’s continuous integration and release processes. In Figure 3, we demonstrate a Tier 1 minified < 100 bytes numeric redundant nested (Tier 1 NRN from [31]) JSON document that consists of an example JSON-e template definition to sort and reverse an array of numbers.

The smallest bit-string produced by JSON BinPack (Schema-driven) (10 bytes) results in a 9% size reduction compared to the next best performing specification: Apache Avro [9] (11 bytes). JSON BinPack (Schema-driven) achieves a 80.7% size reduction compared to the best performing schema-less serialization specifications: JSON BinPack (Schema-less) and MessagePack [10].

Additionally, JSON BinPack (Schema-less) (52 bytes), along with MessagePack [10], produce the smallest bit-string for schema-less binary serialization specifications, resulting in a 1.8% size reduction compared to the next best performing specification: CBOR [3] (53 bytes).

Comparison to Uncompressed and Compressed JSON. In comparison to both JSON [6] (86 bytes) and best-case compressed JSON [6] (86 bytes), JSON BinPack (Schema-driven) (10 bytes) and JSON BinPack (Schema-less) (52 bytes) achieve a 88.3% and 39.5% size reduction, respectively.

---

[12] https://github.com/taskcluster/json-e
[13] https://taskcluster.net
3.3 CircleCI Definition (Blank)

CircleCI \(^{14}\) 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 \(^{15}\). In Figure 4, we demonstrate a Tier 1 minified < 100 bytes numeric non-redundant flat (Tier 1 NNF from [31]) JSON document that represents a simple pipeline configuration file for CircleCI that declares the desired CircleCI version without defining any workflows.

![CircleCI Definition (Blank) benchmark](image)

Figure 4: The space-efficiency benchmark results for the CircleCI Definition (Blank) test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (2 bytes) results in a 50\% size reduction compared to the next best performing specifications: ASN.1 PER Unaligned [23] and Apache Avro [9] (4 bytes). JSON BinPack (Schema-driven) achieves a 80\% size reduction compared to the best performing schema-less serialization specifications: JSON BinPack (Schema-less) and CBOR [3].

Additionally, JSON BinPack (Schema-less) (10 bytes), along with CBOR [3] and MessagePack [10], produce the smallest bit-string for schema-less binary serialization specifications, resulting in a 23\% size reduction compared to the next best performing specification: UBJSON [4] (13 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (14 bytes), JSON BinPack (Schema-driven) (2 bytes) and JSON BinPack (Schema-less) (10 bytes) achieve a 85.7\% and 28.5\% size reduction, respectively. In comparison to best-case compressed JSON [6] (33 bytes), JSON BinPack (Schema-driven) (2 bytes) and JSON BinPack (Schema-less) (10 bytes) achieve a 93.9\% and 69.6\% size reduction, respectively.

\(^{14}\) https://circleci.com

\(^{15}\) https://circleci.com/customers/
3.4 CircleCI Matrix Definition

CircleCI 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. In Figure 5, we demonstrate a Tier 1 minified < 100 bytes numeric non-redundant nested (Tier 1 NNN from [31]) 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.

![CircleCI Matrix Definition](image)

Figure 5: The space-efficiency benchmark results for the CircleCI Matrix Definition test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (7 bytes) results in a 53.3% size reduction compared to the next best performing specification: Apache Avro [9] (15 bytes). JSON BinPack (Schema-driven) achieves a 89.3% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (66 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 8.3% size reduction compared to the next best performing specifications: CBOR [3] and MessagePack [10] (72 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (95 bytes), JSON BinPack (Schema-driven) (7 bytes) and JSON BinPack (Schema-less) (66 bytes) achieve a 92.6% and 30.5% size reduction, respectively. In comparison to best-case compressed JSON [6] (99 bytes), JSON BinPack (Schema-driven) (7 bytes) and JSON BinPack (Schema-less) (66 bytes) achieve a 92.9% and 33.3% size reduction, respectively.

---

16https://circleci.com
17https://circleci.com/customers/
3.5 Grunt.js Clean Task Definition

Grunt.js 18 is an open-source task runner for the JavaScript [7] programming language used by a wide range of companies in the software development industry such as Twitter, Adobe, and Mozilla 19. In Figure 6, we demonstrate a Tier 1 minified < 100 bytes textual redundant flat (Tier 1 TRF from [31]) JSON document that consists of an example configuration for a built-in plugin to clear files and folders called grunt-contrib-clean 20.

![Grunt.js Clean Task Definition](image)

Figure 6: The space-efficiency benchmark results for the Grunt.js Clean Task Definition test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (11 bytes) results in a 15.3% size reduction compared to the next best performing specification: ASN.1 PER Unaligned [23] (13 bytes). JSON BinPack (Schema-driven) achieves a 80.7% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (57 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 5% size reduction compared to the next best performing specification: CBOR [3] and MessagePack [10] (60 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (93 bytes), JSON BinPack (Schema-driven) (11 bytes) and JSON BinPack (Schema-less) (57 bytes) achieve a 88.1% and 38.7% size reduction, respectively. In comparison to best-case compressed JSON [6] (94 bytes), JSON BinPack (Schema-driven) (11 bytes) and JSON BinPack (Schema-less) (57 bytes) achieve a 88.2% and 39.3% size reduction, respectively.

---

18 https://gruntjs.com
19 https://gruntjs.com/who-uses-grunt
20 https://github.com/gruntjs/grunt-contrib-clean
3.6 CommitLint Configuration

CommitLint 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 organization formed by employees from companies including GitHub and Google. In Figure 7, we demonstrate a Tier 1 minified < 100 bytes textual redundant nested (Tier 1 TRN from [31]) 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.

![CommitLint Configuration](image)

Figure 7: The space-efficiency benchmark results for the CommitLint Configuration test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (20 bytes) results in a 50% size reduction compared to the next best performing specification: Apache Avro [9] (40 bytes). JSON BinPack (Schema-driven) achieves a 66.6% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (60 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 18.9% size reduction compared to the next best performing specification: CBOR [3] and MessagePack [10] (74 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (96 bytes), JSON BinPack (Schema-driven) (20 bytes) and JSON BinPack (Schema-less) (60 bytes) achieve a 79.1% and 37.5% size reduction, respectively. In comparison to best-case compressed JSON [6] (80 bytes), JSON BinPack (Schema-driven) (20 bytes) and JSON BinPack (Schema-less) (60 bytes) achieve a 75% and 25% size reduction, respectively.

---

21https://commitlint.js.org/#/
22https://github.com/conventional-changelog
23https://github.com/zeke
24https://github.com/bcoe
3.7 TSLint Linter Definition (Extends Only)

TSLint \(^{25}\) is now an obsolete open-source linter for the TypeScript \(^{26}\) programming language. TSLint was created by the Big Data analytics company Palantir \(^{27}\) and was merged with the ESLint open-source JavaScript linter in 2019 \(^{28}\). In Figure 8, we demonstrate a Tier 1 minified < 100 bytes textual non-redundant flat (Tier 1 TNF from [31]) JSON document that consists of a basic TSLint configuration that only extends a set of existing TSLint configurations.

![Figure 8: The space-efficiency benchmark results for the TSLint Linter Definition (Extends Only) test case selected from the SchemaStore open-source dataset test suite in [31].](image)

The smallest bit-string produced by both JSON BinPack (Schema-driven) and ASN.1 PER Unaligned [23] (46 bytes) results in a 2.1% size reduction compared to the next best performing specifications: Apache Avro [9] and Protocol Buffers [12] (47 bytes). JSON BinPack (Schema-driven) achieves a 16.3% size reduction compared to the best performing schema-less serialization specifications: JSON BinPack (Schema-less), CBOR [3] and MessagePack [10].

Additionally, JSON BinPack (Schema-less) (55 bytes), along with CBOR [3] and MessagePack [10], produce the smallest bit-string for schema-less binary serialization specifications, resulting in a 9.8% size reduction compared to the next best performing specification: Smile [21] (61 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (63 bytes), JSON BinPack (Schema-driven) (46 bytes) and JSON BinPack (Schema-less) (55 bytes) achieve a 26.9% and 12.6% size reduction, respectively. In comparison to best-case compressed JSON [6] (70 bytes), JSON BinPack (Schema-driven) (46 bytes) and JSON BinPack (Schema-less) (55 bytes) achieve a 34.2% and 21.4% size reduction, respectively.

---

\(^{25}\)https://palantir.github.io/tslint

\(^{26}\)https://www.typescriptlang.org

\(^{27}\)https://www.palantir.com

\(^{28}\)https://github.com/palantir/tslint/issues/4534
3.8 ImageOptimizer Azure Webjob Configuration

Image Optimizer\textsuperscript{29} is an Azure App Services WebJob\textsuperscript{30} to compress website images used in the web development industry. In Figure 9, we demonstrate a Tier 1 minified < 100 bytes textual non-redundant nested (Tier 1 TNN from [31]) JSON document that consists of an Image Optimizer configuration to perform lossy compression on images inside a particular folder.

![ImageOptimizer Azure Webjob Configuration](image)

Figure 9: The space-efficiency benchmark results for the ImageOptimizer Azure Webjob Configuration test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by both JSON BinPack (Schema-driven) and ASN.1 PER Un-aligned [23] (21 bytes) results in a 8.6% size reduction compared to the next best performing specification: Protocol Buffers [12] (23 bytes). JSON BinPack (Schema-driven) achieves a 65.5% size reduction compared to the best performing schema-less serialization specifications: JSON BinPack (Schema-less), CBOR [3] and MessagePack [10].

Additionally, JSON BinPack (Schema-less) (61 bytes), along with CBOR [3] and MessagePack [10], produce the smallest bit-string for schema-less binary serialization specifications, resulting in a 12.8% size reduction compared to the next best performing specification: Smile [21] (70 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (82 bytes), JSON BinPack (Schema-driven) (21 bytes) and JSON BinPack (Schema-less) (61 bytes) achieve a 74.3% and 25.6% size reduction, respectively. In comparison to best-case compressed JSON [6] (88 bytes), JSON BinPack (Schema-driven) (21 bytes) and JSON BinPack (Schema-less) (61 bytes) achieve a 76.1% and 30.6% size reduction, respectively.

\textsuperscript{29}https://github.com/madskristensen/ImageOptimizerWebJob

\textsuperscript{30}https://docs.microsoft.com/en-us/azure/app-service/webjobs-create
3.9 SAP Cloud SDK Continuous Delivery Toolkit Configuration

SAP Cloud SDK 31 is a framework that includes support for continuous integration and delivery pipelines to develop applications for the SAP 32 enterprise resource planning platform used by industries such as finance, healthcare and retail. In Figure 10, we demonstrate a Tier 1 minified < 100 bytes boolean redundant flat (Tier 1 BRF from [31]) JSON document that defines a blank pipeline with no declared steps.

![Figure 10: The space-efficiency benchmark results for the SAP Cloud SDK Continuous Delivery Toolkit Configuration test case selected from the SchemaStore open-source dataset test suite in [31].](image)

The smallest bit-string produced by JSON BinPack (Schema-driven), Apache Avro [9] and Protocol Buffers [12] (0 bytes) results in a 100\% size reduction compared to the next best performing specifications: ASN.1 PER Unaligned [23] and Apache Thrift [25] (1 byte).

Additionally, JSON BinPack (Schema-less) (25 bytes), along with CBOR [3] and MessagePack [10], produce the smallest bit-string for schema-less binary serialization specifications, resulting in a 13.7\% size reduction compared to the next best performing specifications: BSON [?] and UBJSON [4] (29 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (44 bytes), JSON BinPack (Schema-driven) (0 bytes) and JSON BinPack (Schema-less) (25 bytes) achieve a 100\% and 43.1\% size reduction, respectively. In comparison to best-case compressed JSON [6] (50 bytes), JSON BinPack (Schema-driven) (0 bytes) and JSON BinPack (Schema-less) (25 bytes) achieve a 100\% and 50\% size reduction, respectively.

---

31 https://sap.github.io/cloud-sdk/
32 https://www.sap.com/index.html
3.10 TSLint Linter Definition (Multi-rule)

TSLint\textsuperscript{33} is now an obsolete open-source linter for the TypeScript\textsuperscript{34} programming language. TSLint was created by the Big Data analytics company Palantir\textsuperscript{35} and was merged with the ESLint open-source JavaScript linter in 2019\textsuperscript{36}. In Figure 11, we demonstrate a Tier 1\textsuperscript{1} minified < 100 bytes boolean redundant nested (Tier 1 BRN from [31]) JSON document that consists of a TSLint configuration that enables and configures a set of built-in rules.

Figure 11: The space-efficiency benchmark results for the TSLint Linter Definition (Multi-rule) test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (1 byte) results in a 75\% size reduction compared to the next best performing specification: ASN.1 PER Unaligned\textsuperscript{23} (4 bytes). JSON BinPack (Schema-driven) achieves a 98.5\% size reduction compared to the best performing schema-less serialization specifications: JSON BinPack (Schema-less), CBOR\textsuperscript{3} and MessagePack\textsuperscript{10}.

Additionally, JSON BinPack (Schema-less) (68 bytes), along with CBOR\textsuperscript{3} and MessagePack\textsuperscript{10}, produce the smallest bit-string for schema-less binary serialization specifications, resulting in a 12.8\% size reduction compared to the next best performing specification: Smile\textsuperscript{21} (78 bytes).

Comparison to Uncompressed and Compressed JSON. In comparison to JSON\textsuperscript{6} (98 bytes), JSON BinPack (Schema-driven) (1 byte) and JSON BinPack (Schema-less) (68 bytes) achieve a 98.9\% and 30.6\% size reduction, respectively. In comparison to best-case compressed JSON\textsuperscript{6} (90 bytes), JSON BinPack (Schema-driven) (1 byte) and JSON BinPack (Schema-less) (68 bytes) achieve a 98.8\% and 24.4\% size reduction, respectively.

\textsuperscript{33}https://palantir.github.io/tslint
\textsuperscript{34}https://www.typescriptlang.org
\textsuperscript{35}https://www.palantir.com
\textsuperscript{36}https://github.com/palantir/tslint/issues/4534
3.11 CommitLint Configuration (Basic)

CommitLint 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 organization formed by employees from companies including GitHub and Google. In Figure 12, we demonstrate a Tier 1 minified < 100 bytes boolean non-redundant flat (Tier 1 BNF from [31]) JSON document that represents a CommitLint configuration file which declares that CommitLint must not use its default commit ignore rules.

![Figure 12: The space-efficiency benchmark results for the CommitLint Configuration (Basic) test case selected from the SchemaStore open-source dataset test suite in [31].](image)

The smallest bit-string produced by both JSON BinPack (Schema-driven) and Protocol Buffers results in a 100% size reduction compared to the next best performing specifications: ASN.1 PER Unaligned and Apache Avro (1 byte).

Additionally, JSON BinPack (Schema-less) (17 bytes), along with CBOR and MessagePack, produce the smallest bit-string for schema-less binary serialization specifications, resulting in a 10.5% size reduction compared to the next best performing specification: UBJSON (19 bytes).

Comparison to Uncompressed and Compressed JSON. In comparison to JSON (25 bytes), JSON BinPack (Schema-driven) (0 bytes) and JSON BinPack (Schema-less) (17 bytes) achieve a 100% and 32% size reduction, respectively. In comparison to best-case compressed JSON (44 bytes), JSON BinPack (Schema-driven) (0 bytes) and JSON BinPack (Schema-less) (17 bytes) achieve a 100% and 61.3% size reduction, respectively.

---

37 https://commitlint.js.org/#/
38 https://github.com/conventional-changelog
39 https://github.com/zeke
40 https://github.com/bcoe
3.12 TSLint Linter Definition (Basic)

TSLint \textsuperscript{41} is now an obsolete open-source linter for the TypeScript \textsuperscript{42} programming language. TSLint was created by the Big Data analytics company Palantir \textsuperscript{43} and was merged with the ESLint open-source JavaScript linter in 2019 \textsuperscript{44}. In Figure 13, we demonstrate a \textbf{Tier 1 minified < 100 bytes boolean non-redundant nested} (Tier 1 BNN from [31]) JSON document that consists of a basic TSLint configuration that enforces grouped alphabetized imports.

![Figure 13: The space-efficiency benchmark results for the TSLint Linter Definition (Basic) test case selected from the SchemaStore open-source dataset test suite in [31].](image)

The smallest bit-string produced by JSON BinPack (Schema-driven), ASN.1 PER Unaligned \textsuperscript{23} and Apache Avro \textsuperscript{9} (1 byte) results in a 87.5\% size reduction compared to the next best performing specifications: Protocol Buffers \textsuperscript{12} and Apache Thrift \textsuperscript{25} (8 bytes). JSON BinPack (Schema-driven) achieves a 98\% size reduction compared to the best performing schema-less serialization specifications: JSON BinPack (Schema-less), CBOR \textsuperscript{3} and MessagePack \textsuperscript{10}.

Additionally, JSON BinPack (Schema-less) (51 bytes), along with CBOR \textsuperscript{3} and MessagePack \textsuperscript{10}, produce the smallest bit-string for schema-less binary serialization specifications, resulting in a 13.5\% size reduction compared to the next best performing specification: Smile \textsuperscript{21} and UBJSON \textsuperscript{4} (59 bytes).

\textbf{Comparison to Uncompressed and Compressed JSON}. In comparison to JSON \textsuperscript{6} (67 bytes), JSON BinPack (Schema-driven) (1 byte) and JSON BinPack (Schema-less) (51 bytes) achieve a 98.5\% and 23.8\% size reduction, respectively. In comparison to best-case compressed JSON \textsuperscript{6} (68 bytes), JSON BinPack (Schema-driven) (1 byte) and JSON BinPack (Schema-less) (51 bytes) achieve a 98.5\% and 25\% size reduction, respectively.

\textsuperscript{41}https://palantir.github.io/tslint
\textsuperscript{42}https://www.typescriptlang.org
\textsuperscript{43}https://www.palantir.com
\textsuperscript{44}https://github.com/palantir/tslint/issues/4534
3.13 GeoJSON Example Document

GeoJSON [5] 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 14, we demonstrate a Tier 2 minified $\geq 100 < 1000$ bytes numeric redundant nested (Tier 2 NRN from [31]) JSON document that defines an example polygon using the GeoJSON format.

![Figure 14: The space-efficiency benchmark results for the GeoJSON Example Document test case selected from the SchemaStore open-source dataset test suite in [31].](image)

The smallest bit-string produced by JSON BinPack (Schema-driven) (82 bytes) results in a 29.9% size reduction compared to the next best performing specification: JSON BinPack (Schema-less) (117 bytes). JSON BinPack (Schema-driven) achieves a 29.9% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (117 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 27.7% size reduction compared to the next best performing specification: MessagePack [10] (162 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (190 bytes), JSON BinPack (Schema-driven) (82 bytes) and JSON BinPack (Schema-less) (117 bytes) achieve a 56.8% and 38.4% size reduction, respectively. In comparison to best-case compressed JSON [6] (116 bytes), JSON BinPack (Schema-driven) (82 bytes) and JSON BinPack (Schema-less) (117 bytes) achieve a 29.3% and negative 0.8% size reduction, respectively.
3.14 OpenWeatherMap API Example Document

OpenWeatherMap is a weather data and forecast API provider used in industries such as energy, agriculture, transportation and construction. In Figure 15, we demonstrate a Tier 2 minified $\geq 100 < 1000$ bytes numeric non-redundant flat (Tier 2 NNF from [31]) JSON document that consists of an HTTP/1.1 [8] response of the weather information in Mountain View, California on June 12, 2019 at 2:44:05 PM GMT.

Figure 15: The space-efficiency benchmark results for the OpenWeatherMap API Example Document test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (113 bytes) results in a 23.6% size reduction compared to the next best performing specification: Apache Avro [9] (148 bytes). JSON BinPack (Schema-driven) achieves a 67.6% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (349 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 8.6% size reduction compared to the next best performing specification: MessagePack [10] (382 bytes).

Comparison to Uncompressed and Compressed JSON. In comparison to JSON [6] (494 bytes), JSON BinPack (Schema-driven) (113 bytes) and JSON BinPack (Schema-less) (349 bytes) achieve a 77.1% and 29.3% size reduction, respectively. In comparison to best-case compressed JSON [6] (341 bytes), JSON BinPack (Schema-driven) (113 bytes) and JSON BinPack (Schema-less) (349 bytes) achieve a 66.8% and negative 2.3% size reduction, respectively.

---

45 https://openweathermap.org
3.15 OpenWeather Road Risk API Example

OpenWeatherMap is a weather data and forecast API provider used in industries such as energy, agriculture, transportation and construction. In Figure 16, we demonstrate a **Tier 2 minified ≥ 100 < 1000 bytes numeric non-redundant nested** (Tier 2 NNN from [31]) JSON document that consists of an example HTTP/1.1 [8] Road Risk API response from the official API documentation that provides weather data and national alerts along a specific route.

![OpenWeather Road Risk API Example](image)

Figure 16: The space-efficiency benchmark results for the OpenWeatherMap Road Risk API Example test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (100 bytes) results in a **35.8%** size reduction compared to the next best performing specification: Apache Avro [9] (156 bytes). JSON BinPack (Schema-driven) achieves a **60.6%** size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (254 bytes) produce the smallest bit-string for schema-less binary serialization specifications, resulting in a **22%** size reduction compared to the next best performing specification: Smile [21] (326 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (375 bytes), JSON BinPack (Schema-driven) (100 bytes) and JSON BinPack (Schema-less) (254 bytes) achieve a **73.3%** and **32.2%** size reduction, respectively. In comparison to best-case compressed JSON [6] (250 bytes), JSON BinPack (Schema-driven) (100 bytes) and JSON BinPack (Schema-less) (254 bytes) achieve a **60%** and negative **1.6%** size reduction, respectively.

---

46 [https://openweathermap.org](https://openweathermap.org)
3.16 TravisCI Notifications Configuration

TravisCI $^{47}$ 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 17, we demonstrate a Tier 2 minified $\geq 100 < 1000$ bytes textual redundant flat (Tier 2 TRF from [31]) 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.

![TravisCI Notifications Configuration](image)

Figure 17: The space-efficiency benchmark results for the TravisCI Notifications Configuration test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (89 bytes) results in a 51.8% size reduction compared to the next best performing specification: JSON BinPack (Schema-less) (185 bytes). JSON BinPack (Schema-driven) achieves a 51.8% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (185 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 18.8% size reduction compared to the next best performing specification: FlexBuffers $^{28}$ (228 bytes).

Comparison to Uncompressed and Compressed JSON. In comparison to JSON $^{6}$ (673 bytes), JSON BinPack (Schema-driven) (89 bytes) and JSON BinPack (Schema-less) (185 bytes) achieve a 86.7% and 72.5% size reduction, respectively. In comparison to best-case compressed JSON $^{6}$ (154 bytes), JSON BinPack (Schema-driven) (89 bytes) and JSON BinPack (Schema-less) (185 bytes) achieve a 42.2% and negative 20.1% size reduction, respectively.

$^{47}$https://travis-ci.com
3.17 Entry Point Regulation Manifest

Entry Point Regulation (EPR) [20] 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 18, we demonstrate a Tier 2 minified $\geq 100 < 1000$ bytes textual redundant nested (Tier 2 TRN from [31]) JSON document that defines an example EPR policy for a fictitious website.

![Graph showing space-efficiency benchmark results for the Entry Point Regulation Manifest test case selected from the SchemaStore open-source dataset test suite in [31].](image)

The smallest bit-string produced by JSON BinPack (Schema-driven) (182 bytes) results in a 6.6% size reduction compared to the next best performing specification: Apache Avro [9] (195 bytes). JSON BinPack (Schema-driven) achieves a 43.3% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (321 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 9.8% size reduction compared to the next best performing specification: Smile [21] (356 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (520 bytes), JSON BinPack (Schema-driven) (182 bytes) and JSON BinPack (Schema-less) (321 bytes) achieve a 65% and 38.2% size reduction, respectively. In comparison to best-case compressed JSON [6] (264 bytes), JSON BinPack (Schema-driven) (182 bytes) and JSON BinPack (Schema-less) (321 bytes) achieve a 31% and negative 21.5% size reduction, respectively.
3.18 JSON Feed Example Document

JSON Feed [24] is a specification for a syndication JSON format similar to RSS [1] and Atom [17] used in the publishing [48] and media [49] industries. In Figure 19, we demonstrate a Tier 2 minified $\geq 100 < 1000$ bytes textual non-redundant flat (Tier 2 TNF from [31]) JSON document that consists of a JSON Feed manifest for an example website that contains a single blog entry.

Figure 19: The space-efficiency benchmark results for the JSON Feed Example Document test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (306 bytes) results in a 23.1% size reduction compared to the next best performing specification: ASN.1 PER Unaligned [23] (398 bytes). JSON BinPack (Schema-driven) achieves a 40.4% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (514 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 0.5% size reduction compared to the next best performing specification: MessagePack [10] (517 bytes).

Comparison to Uncompressed and Compressed JSON. In comparison to JSON [6] (573 bytes), JSON BinPack (Schema-driven) (306 bytes) and JSON BinPack (Schema-less) (514 bytes) achieve a 46.5% and 10.2% size reduction, respectively. In comparison to best-case compressed JSON [6] (327 bytes), JSON BinPack (Schema-driven) (306 bytes) and JSON BinPack (Schema-less) (514 bytes) achieve a 6.4% and negative 57.1% size reduction, respectively.

---

48https://micro.blog
49https://npr.codes/npr-now-supports-json-feed-1c8af29d0ce7
3.19 GitHub Workflow Definition

The GitHub software hosting provider has an automation service called GitHub Actions for projects to define custom workflows. GitHub Actions is used primarily by the open-source software industry. In Figure 20, we demonstrate a Tier 2 minified $\geq 100 < 1000$ bytes textual non-redundant nested (Tier 2 TNN from [31]) JSON document that consists of a simple example workflow definition.

![GitHub Workflow Definition](image)

Figure 20: The space-efficiency benchmark results for the GitHub Workflow Definition test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by both JSON BinPack (Schema-driven) and ASN.1 PER Un-aligned [23] (165 bytes) results in a 1.1% size reduction compared to the next best performing specification: Apache Avro [9] (167 bytes). JSON BinPack (Schema-driven) achieves a 40.4% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (227 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 3.4% size reduction compared to the next best performing specifications: MessagePack [10] and Smile [21] (287 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (356 bytes), JSON BinPack (Schema-driven) (165 bytes) and JSON BinPack (Schema-less) (227 bytes) achieve a 53.6% and 22.1% size reduction, respectively. In comparison to best-case compressed JSON [6] (228 bytes), JSON BinPack (Schema-driven) (165 bytes) and JSON BinPack (Schema-less) (227 bytes) achieve a 27.6% and negative 21.4% size reduction, respectively.

---

[50]https://github.com
[51]https://github.com/features/actions
3.20 GitHub FUNDING Sponsorship Definition (Empty)

The GitHub software hosting provider defines a FUNDING 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 21, we demonstrate a Tier 2 minified \( \geq 100 < 1000 \text{ bytes boolean redundant flat} \) (Tier 2 BRF from [31]) JSON document that consists of a definition that does not declare any supported funding platforms.

![GitHub FUNDING Sponsorship Definition (Empty)](image)

Figure 21: The space-efficiency benchmark results for the GitHub FUNDING Sponsorship Definition (Empty) test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven), ASN.1 PER Unaligned [23] and Apache Avro [9] (16 bytes) results in a 5.8% size reduction compared to the next best performing specification: Protocol Buffers [12] (17 bytes). JSON BinPack (Schema-driven) achieves a 87% size reduction compared to the best performing schema-less serialization specifications: JSON BinPack (Schema-less), CBOR [3] and MessagePack [10].

Additionally, JSON BinPack (Schema-less) (124 bytes), along with CBOR [3] and MessagePack [10], produce the smallest bit-string for schema-less binary serialization specifications, resulting in a 3.8% size reduction compared to the next best performing specifications: Smile [21] (129 bytes).

Comparison to Uncompressed and Compressed JSON. In comparison to JSON [6] (183 bytes), JSON BinPack (Schema-driven) (16 bytes) and JSON BinPack (Schema-less) (124 bytes) achieve a 91.2% and 32.2% size reduction, respectively. In comparison to best-case compressed JSON [6] (134 bytes), JSON BinPack (Schema-driven) (16 bytes) and JSON BinPack (Schema-less) (124 bytes) achieve a 88% and 7.4% size reduction, respectively.

---

52 https://github.com
53 https://docs.github.com/en/github/administering-a-repository/managing-repository-settings/displaying-a-sponsor-button-in-your-repository
### 3.21 ECMAScript Module Loader Definition

`esm` is an open-source ECMAScript [7] module loader for the Node.js [55] 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 22, we demonstrate a **Tier 2 minified ≥ 100 < 1000 bytes boolean non-redundant flat** (Tier 2 BNF from [31]) JSON document that defines an example `esm` configuration.

![ECMAScript Module Loader Definition](image)

**Figure 22:** The space-efficiency benchmark results for the ECMAScript Module Loader Definition test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by both JSON BinPack (Schema-driven) and ASN.1 PER Unaligned [23] (12 bytes) results in a **25%** size reduction compared to the next best performing specification: Apache Avro [9] (16 bytes). JSON BinPack (Schema-driven) achieves a **81.2%** size reduction compared to the best performing schema-less serialization specifications: JSON BinPack (Schema-less), CBOR [3] and MessagePack [10].

Additionally, JSON BinPack (Schema-less) (64 bytes), along with CBOR [3] and MessagePack [10], produce the smallest bit-string for schema-less binary serialization specifications, resulting in a **8.5%** size reduction compared to the next best performing specifications: Smile [21] (70 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (102 bytes), JSON BinPack (Schema-driven) (12 bytes) and JSON BinPack (Schema-less) (64 bytes) achieve a **88.2%** and **37.2%** size reduction, respectively. In comparison to best-case compressed JSON [6] (101 bytes), JSON BinPack (Schema-driven) (12 bytes) and JSON BinPack (Schema-less) (64 bytes) achieve a **88.1%** and **36.6%** size reduction, respectively.

---

54 https://github.com/standard-things/esm  
55 https://nodejs.org
3.22 ESLint Configuration Document

ESLint \[56\] is a popular open-source extensible linter for the JavaScript \[7\] programming language used by a wide range of companies in the software development industry such as Google, Salesforce, and Airbnb. In Figure 23, we demonstrate a Tier 3 minified \(\geq 1000\) bytes numeric redundant flat (Tier 3 NRF from \[31\]) JSON document that defines a browser and Node.js linter configuration that defines general-purposes and React.js-specific \[57\] linting rules.

![ESLint Configuration Document](image)

**Figure 23:** The space-efficiency benchmark results for the ESLint Configuration Document test case selected from the SchemaStore open-source dataset test suite in \[31\].

The smallest bit-string produced by JSON BinPack (Schema-driven) (64 bytes) results in a 1.5\% size reduction compared to the next best performing specification: ASN.1 PER Unaligned \[23\] (65 bytes). JSON BinPack (Schema-driven) achieves a 93.3\% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (969 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 0.2\% size reduction compared to the next best performing specifications: MessagePack \[10\] (971 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON \[6\] (1141 bytes), JSON BinPack (Schema-driven) (64 bytes) and JSON BinPack (Schema-less) (969 bytes) achieve a 94.3\% and 15\% size reduction, respectively. In comparison to best-case compressed JSON \[6\] (488 bytes), JSON BinPack (Schema-driven) (64 bytes) and JSON BinPack (Schema-less) (969 bytes) achieve a 86.8\% and negative 98.5\% size reduction, respectively.

---

\[56\]https://eslint.org
\[57\]https://reactjs.org
3.23 NPM Package.json Linter Configuration Manifest

Node.js Package Manager (NPM) \(^{58}\) is an open-source package manager for Node.js \(^{39}\), a JavaScript \(^{7}\) runtime targeted at the web development industry. npm-package-json-lint \(^{60}\) is an open-source tool to enforce a set of configurable rules for a Node.js Package Manager (NPM) \(^{61}\) configuration manifest. In Figure 24, we demonstrate a **Tier 3 minified ≥ 1000 bytes textual redundant flat** (Tier 3 TRF from [31]) JSON document that consists of an example npm-package-json-lint configuration.

![NPM Package.json Linter Configuration Manifest](image)

Figure 24: The space-efficiency benchmark results for the NPM Package.json Linter Configuration Manifest test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (90 bytes) results in a **55.2%** size reduction compared to the next best performing specification: Apache Avro \(^{9}\) (201 bytes). JSON BinPack (Schema-driven) achieves a **88.6%** size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (791 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a **16%** size reduction compared to the next best performing specifications: FlexBuffers \(^{28}\) (942 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON \(^{6}\) (1159 bytes), JSON BinPack (Schema-driven) (90 bytes) and JSON BinPack (Schema-less) (791 bytes) achieve a **92.2%** and **31.7%** size reduction, respectively. In comparison to best-case compressed JSON \(^{6}\) (321 bytes), JSON BinPack (Schema-driven) (90 bytes) and JSON BinPack (Schema-less) (791 bytes) achieve a **71.9%** and negative **146.4%** size reduction, respectively.

---

\(^{58}\)https://www.npmjs.com
\(^{59}\)https://nodejs.org
\(^{60}\)https://npmpackagejsonlint.org/en/
\(^{61}\)https://www.npmjs.com

29
3.24 .NET Core Project

The ASP.NET 62 Microsoft web-application framework defined a now-obsolete JSON-based project manifest called project.json 63 used in the web industry. In Figure 25, we demonstrate a Tier 3 minified $\geq 1000$ bytes textual redundant nested JSON document that consists of a detailed example project.json manifest that lists several dependencies.

![Figure 25: The space-efficiency benchmark results for the .NET Core Project test case selected from the SchemaStore open-source dataset test suite in [31].](image)

The smallest bit-string produced by JSON BinPack (Schema-driven) (132 bytes) results in a 45.4% size reduction compared to the next best performing specification: ASN.1 PER Unaligned [23] and Apache Avro [9] (242 bytes). JSON BinPack (Schema-driven) achieves a 82.3% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (748 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 14% size reduction compared to the next best performing specifications: Smile [21] (870 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (1049 bytes), JSON BinPack (Schema-driven) (132 bytes) and JSON BinPack (Schema-less) (748 bytes) achieve a 87.4% and 28.6% size reduction, respectively. In comparison to best-case compressed JSON [6] (411 bytes), JSON BinPack (Schema-driven) (132 bytes) and JSON BinPack (Schema-less) (748 bytes) achieve a 67.8% and negative 81.9% size reduction, respectively.

---

62https://dotnet.microsoft.com/apps/aspnet

63http://web.archive.org/web/20150322033428/https://github.com/aspnet/Home/wiki/Project.json-file
3.25 NPM Package.json Example Manifest

Node.js Package Manager (NPM) is an open-source package manager for Node.js, a JavaScript runtime targeted at the web development industry. A package that is published to NPM is declared using a JSON file called package.json. In Figure 26, we demonstrate a Tier 3 minified ≥ 1000 bytes textual non-redundant flat (Tier 3 TNF from [31]) JSON document that consists of a package.json manifest that declares a particular version of the Grunt.js task runner.

![NPM Package.json Example Manifest](image)

Figure 26: The space-efficiency benchmark results for the NPM Package.json Example Manifest test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (947 bytes) results in a 36.7% size reduction compared to the next best performing specification: ASN.1 PER Unaligned [23] (1498 bytes). JSON BinPack (Schema-driven) achieves a 51.6% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (1957 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 1.3% size reduction compared to the next best performing specifications: Smile [21] (1983 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (2259 bytes), JSON BinPack (Schema-driven) (947 bytes) and JSON BinPack (Schema-less) (1957 bytes) achieve a 58% and 13.3% size reduction, respectively. In comparison to best-case compressed JSON [6] (1093 bytes), JSON BinPack (Schema-driven) (947 bytes) and JSON BinPack (Schema-less) (1957 bytes) achieve a 13.3% and negative 79% size reduction, respectively.

---

64 [https://www.npmjs.com](https://www.npmjs.com)
65 [https://nodejs.org](https://nodejs.org)
66 [https://docs.npmjs.com/cli/v6/configuring-npm/package-json](https://docs.npmjs.com/cli/v6/configuring-npm/package-json)
67 [https://gruntjs.com](https://gruntjs.com)
3.26 JSON Resume Example

JSON Resume \(^{68}\) is a community-driven proposal for a JSON-based file format that declares and renders themable resumes used in the recruitment industry. In Figure 27, we demonstrate a **Tier 3 minified \(\geq 1000\) bytes textual non-redundant nested** (Tier 3 TNN from [31]) JSON document that consists of a detailed example resume for a fictitious software programmer.

Figure 27: The space-efficiency benchmark results for the JSON Resume Example test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (1468 bytes) results in a **31.4%** size reduction compared to the next best performing specification: ASN.1 PER Unaligned [23] (2143 bytes). JSON BinPack (Schema-driven) achieves a **43.9%** size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (2619 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a **0.03%** size reduction compared to the next best performing specifications: Smile [21] (2620 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (3048 bytes), JSON BinPack (Schema-driven) (1468 bytes) and JSON BinPack (Schema-less) (2619 bytes) achieve a **51.8%** and **14%** size reduction, respectively. In comparison to best-case compressed JSON [6] (1556 bytes), JSON BinPack (Schema-driven) (1468 bytes) and JSON BinPack (Schema-less) (2619 bytes) achieve a **5.6%** and negative **68.3%** size reduction, respectively.

---

\(^{68}\)https://jsonresume.org
3.27 Nightwatch.js Test Framework Configuration

Nightwatch.js is an open-source browser automation solution used in the software testing industry. In Figure 28, we demonstrate a Tier 3 minified ≥ 1000 bytes boolean redundant flat (Tier 3 BRF from [31]) JSON document that consists of a Nightwatch.js configuration file that defines a set of general-purpose WebDriver and Selenium options.

![Nightwatch.js Test Framework Configuration](image)

Figure 28: The space-efficiency benchmark results for the Nightwatch.js Test Framework Configuration test case selected from the SchemaStore open-source dataset test suite in [31].

The smallest bit-string produced by JSON BinPack (Schema-driven) (73 bytes) results in a 17.9% size reduction compared to the next best performing specification: ASN.1 PER Unaligned [23] (89 bytes). JSON BinPack (Schema-driven) achieves a 93.2% size reduction compared to the best performing schema-less serialization specification: JSON BinPack (Schema-less).

Additionally, JSON BinPack (Schema-less) (1085 bytes) produces the smallest bit-string for schema-less binary serialization specifications, resulting in a 0.4% size reduction compared to the next best performing specifications: Smile [21] (1090 bytes).

**Comparison to Uncompressed and Compressed JSON.** In comparison to JSON [6] (1507 bytes), JSON BinPack (Schema-driven) (73 bytes) and JSON BinPack (Schema-less) (1085 bytes) achieve a 95.1% and 28% size reduction, respectively. In comparison to best-case compressed JSON [6] (649 bytes), JSON BinPack (Schema-driven) (73 bytes) and JSON BinPack (Schema-less) (1085 bytes) achieve a 88.7% and negative 67.1% size reduction, respectively.

3.28 Reproducibility

To make our pre-production benchmark reproducible, we followed the reproducibility levels introduced by [13]. We aimed for Level 3, the highest-level of reproducibility, as justified in the following sections.

The pre-production implementation of JSON BinPack is written using the TypeScript programming language. It contains valid encoding rules, mapping rules and canonicalization rules but it is slow and memory-consuming compared to a production-ready implementation.

- **Supported Environments for the Pre-production JSON BinPack Implementation:**
  The pre-production JSON BinPack implementation and benchmark software are written 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 do expect the

---

[^69]: https://nightwatchjs.org
[^70]: https://www.selenium.dev
benchmark software to run on an msys2 71 or Windows Subsystem for Linux 72 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 would not affect the results of the benchmark. No further conditions apart from the exact software versions of the dependencies required by the project are necessary to replicate the results.

- **Automation.** The benchmark software, from the generation of the serialized bit-strings to the generation of the plots using Matplotlib 73, is automated through a GNU Make 74 declarative and parallelizable build definition.

- **Testing.** The POSIX shell and Python scripts distributed with the benchmark are automatically linted using the shellcheck 75 and flake8 76 open-source tools, respectively. The serialization and deserialization procedures of the benchmark are automatically tested as explained in [31].

- **Documentation and Readability.** The README file 77 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 successfully 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 [30]. The DOI includes the source code for reproducing the benchmark and the associated results.

- **Dependencies.** The benchmark software is implemented using established open-source software with the exception of the ASN-1Step 78 command-line tool, which is a proprietary implementation of ASN.1 [22] 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 [23] binary serialization specification.

- **Version Control.** The benchmark repository utilises the git 79 version control system and its publicly hosted on GitHub 80 as recommended by [19].

- **Continuous Integration.** The GitHub repository hosting the benchmark software is setup with the GitHub Actions 81 continuous integration provider 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 82 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 [6] documents being encoded by the benchmark and direct

---

71 https://www.msys2.org
72 https://docs.microsoft.com/en-us/windows/wsl/
73 https://matplotlib.org
74 https://www.gnu.org/software/make/
75 https://www.shellcheck.net
76 https://flake8.pycqa.org/
77 https://github.com/jviotti(binary-json-size-benchmark#running-locally
78 https://www.oss.com/asn1/products/asn-1step/asn-1step.html
79 https://git-scm.com
80 https://github.com/jviotti(binary-json-size-benchmark
81 https://github.com/features/actions
82 https://www.apache.org/licenses/LICENSE-2.0.html
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 committed to accepting open-source contributions.

4 Conclusions

In this section, we critically evaluate the JSON BinPack pre-production implementation against schema-driven and schema-less alternative binary serialization specifications, motivating our discussion starting with the summary in Table 4.

Table 4: A summary of the size reduction provided by the JSON BinPack binary serialization specification proof-of-concept implementation in both schema-driven and schema-less mode in comparison to JSON [6] given the input data listed in [31].

| Mode                    | Size Reductions in Comparison To JSON | Negative Cases |
|-------------------------|---------------------------------------|----------------|
|                         | Maximum | Minimum | Range | Median | Average |                |
| JSON BinPack (Schema-driven) | 100%    | 26.9%   | 73    | 86.7%  | 78.7%   | 0 / 27 (0%)  |
| JSON BinPack (Schema-less) | 72.5%   | 10.2%   | 62.2  | 30.6%  | 30.5%   | 0 / 27 (0%)  |
| Averages                | 86.2%   | 18.6%   | 67.6  | 58.6%  | 54.6%   | 0%            |

4.1 Q1: How does JSON BinPack in schema-driven mode compare to JSON in terms of space-efficiency?

As demonstrated in section 3 and Table 5, the JSON BinPack schema-driven binary serialization specification, denoted as JSON BinPack (Schema-driven), is comparatively as or more space-efficient than every other serialization specification for the 27 proposed input data considered by [31]. Unlike any other considered binary serialization specification, JSON BinPack is strictly space-efficient in comparison to JSON [6] given the input data.

In [31], we found that the most space-efficient JSON-compatible binary serialization specifications are ASN.1 PER Unaligned [23] and Apache Avro [9]. ASN.1 PER Unaligned [23] results in 71.4% and 65.7% median and average size reductions with a maximum of 98.5% and a minimum of negative 7.9%. Apache Avro [9] results in 73.5% and 65.7% median and average size reductions with a maximum of 100% and a minimum of negative 48.9%. In comparison, JSON BinPack produces strictly space-efficient results with a smaller range and no negative cases: 86.7% and 78.7% median and average size reductions with a maximum of 100% and a minimum of 26.9% as shown in Table 4. Additionally, JSON BinPack provides improvements in terms of space-efficiency in comparison to the best-performing schema-driven binary serialization specifications for documents that are highly redundant or nested according to the JSON [6] taxonomy introduced in [31]. For example, JSON BinPack produces bit-strings that are 82%, 75% and 60% smaller than the best-performing schema-driven alternatives for TravisCI Notifications Configuration (see subsection 3.16), TSLint Linter Definition (Multi-rule) (see subsection 3.10) and GeoJSON Example Document (see subsection 3.13), respectively.

JSON BinPack matches but not increases the size reduction characteristics provided by the most space-efficient serialization specifications in 8 out of the 27 cases. In terms of the taxonomy for JSON [6] documents introduced in [31], this list includes 5 out of the 7 documents considered boolean and 3 out of the 6 documents considered textual and non-redundant. These 8 documents represent cases where ASN.1 PER Unaligned [23], Apache Avro [9] and Protocol Buffers [12] perform at or close to the optimal level in terms of space-efficiency. For example, JSON BinPack
and Protocol Buffers [12], and JSON BinPack, Apache Avro [9] and Protocol Buffers [12], serialize the CommitLint Configuration (Basic) Tier 1 Minified < 100 bytes, boolean, non-redundant and flat (see subsection 3.11) and the SAP Cloud SDK Continuous Delivery Toolkit Configuration Tier 1 Minified < 100 bytes, boolean, redundant and flat (see subsection 3.9) JSON [6] documents, respectively, into 0-byte bit-strings.

Table 5: A summary of the top-performing and second top-performing schema-driven serialization specifications for every document considered in [31]. Percentages represent the size reduction achieved in comparison to JSON. More is better.

| Document Name                                           | Best Performing Specifications | Second Best Performing Specifications |
|---------------------------------------------------------|--------------------------------|---------------------------------------|
| JSON-e Templating Engine Sort Example                   | JSON BinPack (76.4%)          | Apache Avro (73.5%)                   |
| JSON-e Templating Engine Reverse Sort Example           | JSON BinPack (88.3%)          | Apache Avro (87.2%)                   |
| CircleCI Definition (Blank)                             | JSON BinPack (85.7%)          | ASN.1 and Apache Avro (71.4%)         |
| CircleCI Matrix Definition                              | JSON BinPack (92.6%)          | Apache Avro (84.2%)                   |
| Grunt.js Clean Task Definition                          | JSON BinPack (88.1%)          | ASN.1 (86%)                           |
| CommitLint Configuration                               | JSON BinPack (79.1%)          | Apache Avro (58.3%)                   |
| TSLint Linter Definition (Extends Only)                 | JSON BinPack and ASN.1 (26.9%)| Apache Avro and Protocol Buffers (25.3%)|
| ImageOptimizer Azure Webjob Configuration               | JSON BinPack and ASN.1 (74.3%)| Protocol Buffers (71.9%)              |
| SAP Cloud SDK Continuous Delivery Toolkit Configuration | JSON BinPack, Apache Avro and Protocol Buffers (100%) | ASN.1 and Apache Thrift (97.7%) |
| TSLint Linter Definition (Multi-rule)                   | JSON BinPack (98.9%)          | ASN.1 (95.9%)                         |
| CommitLint Configuration (Basic)                        | JSON BinPack and Protocol Buffers (100%) | ASN.1 and Apache Avro (96%) |
| TSLint Linter Definition (Basic)                        | JSON BinPack, ASN.1 and Apache Avro (98.5%) | Protocol Buffers and Apache Thrift (88%) |
| GeoJSON Example Document                                | JSON BinPack (56.8%)          | ASN.1 (-7.8%)                         |
| OpenWeatherMap API Example Document                     | JSON BinPack (77.1%)          | Apache Avro (70%)                     |
| OpenWeather Road Risk API Example                       | JSON BinPack (73.3%)          | Apache Avro (58.4%)                   |
| TravisCI Notifications Configuration                    | JSON BinPack (86.7%)          | ASN.1 (26.1%)                         |
| Entry Point Regulation Manifest                         | JSON BinPack (65%)            | Apache Avro (62.5%)                   |
| JSON Feed Example Document                              | JSON BinPack (46.5%)          | ASN.1 (30.5%)                         |
| GitHub Workflow Definition                              | JSON BinPack and ASN.1 (53.6%)| Apache Avro (53%)                     |
| GitHub FUNDING Sponsorship Definition (Empty)           | JSON BinPack, ASN.1 and Apache Avro (91.2%) | Protocol Buffers (90.7%) |
| ECMAScript Module Loader Definition                     | JSON BinPack and ASN.1 (88.2%)| Apache Avro (84.3%)                   |
| ESLint Configuration Document                            | JSON BinPack (94.3%)          | ASN.1 (94.3%)                         |
| NPM Package.json Linter Configuration Manifest          | JSON BinPack (92.2%)          | Apache Avro (82.6%)                   |
| .NET Core Project                                       | JSON BinPack (87.4%)          | ASN.1 and Apache Avro (76.9%)         |
| NPM Package.json Example Manifest                       | JSON BinPack (58%)            | ASN.1 (33.6%)                         |
| JSON Resume Example                                      | JSON BinPack (51.8%)          | ASN.1 (29.6%)                         |
| Nightwatch.js Test Framework Configuration              | JSON BinPack (95.1%)          | ASN.1 (94%)                           |
4.2 Q2: How does JSON BinPack in schema-less mode compare to JSON in terms of space-efficiency?

As explained in section 2, we benchmark JSON BinPack in schema-less mode by serializing every input JSON [6] document with a loose JSON Schema [35] definition that matches every instance. As shown in section 3 and Table 6, the schema-less mode of the JSON BinPack binary serialization specification, denoted as JSON BinPack (Schema-less), is as space-efficient or more space-efficient than every other schema-less serialization specification considered by [31] for the 27 proposed input data. Like CBOR [3] and MessagePack [10], JSON BinPack in schema-less mode is strictly space-efficient in comparison to JSON [6]. However, JSON BinPack in schema-driven mode is strictly space-efficient in comparison to JSON BinPack in schema-less mode.

In [31], we find that the most space-efficient JSON-compatible binary schema-less serialization specifications are CBOR [3] and MessagePack [10]. CBOR [3] results in 22.5% and 22.4% median and average size reductions with a maximum of 43.2% and a minimum of 6.8%. Similarly, MessagePack [10] results in 22.7% and 22.8% median and average size reductions with a maximum of 43.2% and a minimum of 6.8%. In comparison, JSON BinPack in schema-less mode produces strictly space-efficient results: 30.6% and 30.5% median and average size reductions with a maximum of 72.5% and a minimum of 10.2% as shown in Table 4. Additionally, JSON BinPack in schema-less mode provides significant improvements in terms of space-efficiency in comparison to the best-performing schema-less binary serialization specifications for documents that have a high-degree of nesting according to the JSON [6] taxonomy defined in [31]. For example, JSON BinPack produces bit-strings that are 27.7%, 22% and 18.9% smaller than the best-performing schema-less alternatives for GeoJSON Example Document (see subsection 3.13), OpenWeather Road Risk API Example (see subsection 3.15) and CommitLint Configuration (see subsection 3.6), respectively. Furthermore, JSON BinPack in schema-less mode produces space-efficient results in comparison to every schema-driven binary serialization specification for GeoJSON Example Document (see subsection 3.13) and TravisCI Notifications Configuration (see subsection 3.16), only second to JSON BinPack executed in schema-driven mode.

JSON BinPack in schema-less mode matches but does not increase the size reduction characteristics provided by the most space-efficient schema-less serialization specifications in 11 out of 27 cases. In terms of the taxonomy for JSON [6] documents defined in [31], this list includes 9 out of the 12 documents considered Tier 1 Minified < 100 bytes and 2 out of the 2 documents considered Tier 2 Minified ≥ 100 < 1000 bytes and boolean. These 11 documents represent cases where both CBOR [3] and MessagePack [10] perform close to the optimal level in terms of space-efficiency for a schema-less serialization specification.

4.3 Q3: How does JSON BinPack in schema-driven and schema-less mode compare to compressed JSON?

In [31], we conclude that general-purpose data compression tends to yield negative results for JSON [6] documents that are Tier 1 Minified < 100 bytes according to the proposed taxonomy given that the auxiliary data structures encoded by dictionary-based compressors may exceed the size of such small input documents. However, leaving Tier 1 Minified < 100 bytes documents aside, best-case compressed JSON [6] is space-efficient in comparison to the considered schema-less binary serialization specifications in 86.6% of the cases. Leaving Tier 1 Minified < 100 bytes documents aside, best-case compressed JSON [6] is strictly space-efficiency in comparison to the considered schema-driven binary serialization specifications in 33.3% of the cases.

While JSON BinPack in schema-less mode matches or outperforms the alternative schema-less binary serialization specifications considered in [31] as shown in subsection 4.2, best-case compressed JSON [6] is space-efficient in comparison to JSON BinPack in schema-less mode in 13 out of the 27 considered cases as shown in Table 7. Of these 13 negative cases, 8 documents are considered textual according to the taxonomy defined in [31]. However, unlike the other considered schema-driven binary serialization specifications, JSON BinPack in schema-driven mode is space-efficient in comparison to best-case compressed JSON [6] as shown in Table 7 in terms of the median and average with size reductions of 76.1% and 66.8%, respectively. Existing literature [11] [2] show that compressed textual schema-less serialization specifications such as JSON [6] can outperform compressed and uncompressed schema-driven binary serialization specifications in terms of space-
Table 6: A summary of the top-performing and second top-performing schema-less serialization specifications for every document considered in [31]. Percentages represent the size reduction achieved in comparison to JSON. More is better.

| Document Name                        | Best Performing Specifications | Second Best Performing Specifications |
|--------------------------------------|-------------------------------|---------------------------------------|
| JSON-e Templating Engine Sort Example| JSON BinPack, CBOR and MessagePack (38.2%) | Smile (20.5%) |
| JSON-e Templating Engine Reverse Sort Example| JSON BinPack and MessagePack (39.5%) | CBOR (38.3%) |
| CircleCI Definition (Blank)          | JSON BinPack, CBOR and MessagePack (28.5%) | UBJSON (7.1%) |
| CircleCI Matrix Definition           | JSON BinPack (30.5%)           | CBOR and MessagePack (24.2%) |
| Grunt.js Clean Task Definition       | JSON BinPack (38.7%)           | CBOR and MessagePack (35.4%) |
| CommitLint Configuration             | JSON BinPack (37.5%)           | CBOR and MessagePack (22.9%) |
| TSLint Linter Definition (Extends Only) | JSON BinPack, CBOR and MessagePack (12.6%) | Smile (3.1%) |
| ImageOptimizer Azure Webjob Configuation | JSON BinPack, CBOR and MessagePack (25.6%) | Smile (14.6%) |
| SAP Cloud SDK Continuous Delivery Toolkit Configuration | JSON BinPack, CBOR and MessagePack (43.1%) | BSON and UBJSON (34%) |
| TSLint Linter Definition (Multi-rule) | JSON BinPack, CBOR and MessagePack (30.6%) | Smile (20.4%) |
| CommitLint Configuration (Basic)     | JSON BinPack, CBOR and MessagePack (32%) | UBJSON (24%) |
| TSLint Linter Definition (Basic)     | JSON BinPack, CBOR and MessagePack (23.8%) | Smile and UBJSON (11.9%) |
| GeoJSON Example Document             | JSON BinPack (38.4%)           | MessagePack (14.7%) |
| OpenWeatherMap API Example Document  | JSON BinPack (29.3%)           | MessagePack (22.6%) |
| OpenWeather Road Risk API Example    | JSON BinPack (32.2%)           | Smile (13%) |
| TravisCI Notifications Configuration | JSON BinPack (72.5%)           | FlexBuffers (66.1%) |
| Entry Point Regulation Manifest      | JSON BinPack (38.2%)           | Smile (31.5%) |
| JSON Feed Example Document           | JSON BinPack (10.2%)           | MessagePack (9.7%) |
| GitHub Workflow Definition           | JSON BinPack (22.1%)           | MessagePack and Smile (19.3%) |
| GitHub FUNDING Sponsorship Definition (Empty) | JSON BinPack, CBOR and MessagePack (32.2%) | Smile (29.5%) |
| ECMAScript Module Loader Definition  | JSON BinPack, CBOR and MessagePack (37.2%) | Smile (31.3%) |
| ESLint Configuration Document        | JSON BinPack (15%)             | MessagePack (14.8%) |
| NPM Package.json Linter Configuration Manifest | JSON BinPack (31.7%) | FlexBuffers (18.7%) |
| .NET Core Project                    | JSON BinPack (28.6%)           | Smile (17%) |
| NPM Package.json Example Manifest    | JSON BinPack (13.3%)           | Smile (12.2%) |
| JSON Resume Example                  | JSON BinPack (14.07%)          | Smile (14.04%) |
| Nightwatch.js Test Framework         | JSON BinPack (28%)             | Smile (27.6%) |

Efficiency. However, we conclude that a space-efficient schema-driven serialization specification such as JSON BinPack can outperform general-purpose data compression.
Table 7: A summary of the size reduction provided by the JSON BinPack binary serialization specification pre-production implementation in both schema-driven and schema-less mode in comparison to the best case scenarios of compressed JSON [6] given the compression formats and input data listed in [31].

| Mode                  | Size Reductions in Comparison To Compressed JSON | Negative Cases |
|-----------------------|--------------------------------------------------|----------------|
|                       | Maximum      | Minimum | Range | Median | Average |                   |
| JSON BinPack (Schema-driven) | 100%        | 5.65%   | 94.3  | 76.1%  | 66.8%   | 0 / 27 (0%)       |
| JSON BinPack (Schema-less)  | 69.6%       | -146.4% | 216.1 | 7.4%   | -5.27%  | 13 / 27 (48.1%)   |
| Averages              | 84.4%       | -70.3%  | 155.2 | 41.7%  | 30.7%   | 24%               |

5 Future Work

On-going JSON Schema Support in JSON BinPack. The pre-production JSON BinPack implementation benchmark in this paper does not support every keyword defined by the JSON Schema Core [35] and Validation [36] specifications. Every JSON Schema document is supported by JSON BinPack by definition, as JSON BinPack is designed to gracefully fallback to a schema-less encoding when encountering unrecognized keywords. However, explicit support for JSON Schema keywords increases space-efficiency when such keywords are in use. We intend to continue writing encodings, canonicalization and mapping rules to cover every official JSON Schema vocabulary.

Production-ready JSON BinPack Implementation. This study considers a pre-production implementation of JSON BinPack written using the TypeScript programming language to demonstrate the space-efficiency potential of the proposed serialization specification. To make JSON BinPack suitable for production usage, we intend to produce a new implementation using a systems programming language.

Runtime Efficiency Benchmark. Following the development of a production-ready implementation of JSON BinPack written using a systems programming language, we hope to pursue a runtime-efficiency benchmark that looks the characteristics such as serialization and deserialization speed, and memory consumption.

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] Edman Anjos, Junhee Lee, and Srinivasa Rao Satti. 2016. SJSON: A succinct representation for JavaScript object notation documents. , 173–178 pages.

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

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

[5] 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

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

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

[8] R. Fielding and J. Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC. https://doi.org/10.17487/RFC7231
[9] The Apache Software Foundation. 2012. *Apache Avro™ 1.10.0 Specification*. The Apache Software Foundation. https://avro.apache.org/docs/current/spec.html

[10] 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

[11] 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

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

[13] 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

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

[15] 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

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

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

[18] Jean Paoli, François Yergeau, Tim Bray, Eve Maler, and Michael Sperberg-McQueen. 2006. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C Recommendation. W3C. https://www.w3.org/TR/2006/REC-xml-20060816/.

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

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

[21] Tatu Saloranta. 2017. Efficient JSON-compatible binary format: “Smile”. FasterXML. https://github.com/FasterXML/smile-format-specification/blob/master/smile-format-specification.md

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

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

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

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

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

[27] Wouter van Oortmerssen. 2014. *FlatBuffers: Writing a Schema*. Google. https://google.github.io/flatbuffers/flatbuffers_guide_writing_schema.html
[28] Wouter van Oortmerssen. 2017. *FlexBuffers*. Google. https://google.github.io/flatbuffers/flexbuffers.html

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

[30] Juan Cruz Viotti. 2022. *jviotti/binary-json-size-benchmark: v3.0.0*. https://doi.org/10.5281/zenodo.7076479

[31] Juan Cruz Viotti and Mital Kinderkhedia. 2022. A Benchmark of JSON-compatible Binary Serialization Specifications. arXiv:2201.03051 [cs.SE]

[32] Juan Cruz Viotti and Mital Kinderkhedia. 2022. JSON Stats Analyzer. https://doi.org/10.48550/ARXIV.2211.11314

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

[34] 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

[35] 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

[36] A. Wright, H. Andrews, and B. Hutton. 2021. *JSON Schema: A Media Type for Describing JSON Documents*. Technical Report. IETF. https://tools.ietf.org/html/draft-bhutton-json-schema-validation-00