Maldives Journal of Engineering and Technology
ISSN: 3007-6536 (Online)
https://mjet.mnu.edu.mv/

The Impact of Binary Serialization On Lottie Files: A Comparative Study

Iyaan Azeez and Ibrahim Thorig

Department of Computer Science, Faculty of Engineering, Science and Technology, The Maldives National University, Maldives;

**Corresponding: s082605@student.mnu.edu.mv;*

Abstract: As interactive web animations become more essential to modern user interface, the demand for more scale-able animation formats increases. the standard Lottie format relies on human-readable JSON metadata for asset delivery. However, text-based serialization imposes a severe computational parsing tax on client devices, creating performance bottlenecks. This study addresses the lack of empirical data regarding a specialized, schema-driven binary serialization format tailored specifically to enhance the transfer and loading of Lottie animations. we developed an alternative binary format prototype, TINY-LOTTIE, which structures data layout to align directly with modern machine architectures. Benchmarking across a structurally diverse dataset ($N = 9$) reveals that it was able to achieve 7.46 X size reduction compared to its original JSON alternative. Moreover, this was accomplished while keeping the encoding time under 45ms range. The average encoding time reduction is about 80.3 X compared to *Dot-Lottie*. These findings demonstrate that shifting the optimization paradigm from general stream compression to structural binary formatting unlocks deterministic scalability and enables low-level CPU parallelization. This research palliates a critical gap between network transmission budgets and client-side computational latency, establishing a foundational framework for next-generation, high-performance multimedia assets.

Keywords: Multimedia; Compression; Serialization; JSON; Schema-based

1. INTRODUCTION

The modern web has changed drastically over the last decade. It has shifted from static websites with minimal interactivity to more fluid and interactive applications representing full platforms. This evolution has been largely driven by technologies that allow designers to export complex animations from professional tools into the web platforms. The primary format that is facilitating the widespread adoption of animations is Lottie. It uses JSON as its base serialization format which allows it to be seamlessly transmitted across the web platform. There are multitude of reasons why JSON as a serialization format is not suitable for file format with such dense data such as Lottie Files.

Received: 19 February 2026
Accepted: 5 May 2026
Published: 28 May 2026

- *Storage Inefficiency* – JSON is a textual encoding, meaning numbers are not represented in its native form, which scales poorly with large arrays if numerical shapes and layer data characteristics of Lottie files.
- *Parser Complexity* – A JSON parser need to perform expensive guesswork to identify field types during decoding. A decoder for JSON data is more computationally expensive as you cannot sequentially go through the file
- *Cannot Parallelize Effectively* – The lack of a fixed-size offset or index structure in JSON makes lexical analysis difficult to parallelize; a parser must typically process the byte stream sequentially, which limits the potential of modern multi-core processors.

The Gap in Current Research: A lot of research has been done in the field of creating JSON compatible binary serialization format that improves upon it. They are mostly designed as a general format built to support various use cases. There are almost no studies done on how we can fuse ideas of binary serialization into *Lottie*. There are tools in the industry that explored how to shrink files over the wire, but they heavily neglect the computational tax that decompression and parsing impose on mobile or low-power client devices immediately before rendering begins. By providing comparative benchmarks detailing the structural variance of animations this paper supplies missing data needed to construct predictable performance budgets.

There are more general binary serialization formats such as *Protocol Buffer* [1] and *Flat Buffers* [2]. They allow a general schema to be written which can then be used to turn any structure from a programming language into a bit string. They have a set of common data structures which allow different types of data to be serialized. They are often designed for diverse use cases and may not be optimized for the specific hierarchical and data-intensive nature of animation files. Consequently, there is a lack of data regarding a specialized, compact binary format tailored specifically to enhance the transmission and unpacking speed of Lottie animations in performance-critical or low-bandwidth environments. In [3], [4], [5] they come to the conclusion that an immediate improvement in payload efficiency and response times were achieved when switching to a binary format compared to a JSON format when sending data through a network. Previous work by [6] has attempted to minimize the size of Lottie files by using G-zip as a container and the deflate algorithm as a general compressor.

In their comprehensive survey [7] distinguished between schema-less and schema-driven frameworks. Schema-less formats retain key-value metadata within the payload, offering flexibility but failing to completely resolve size inflation. Conversely, schema-driven protocols isolate structural definitions from the data entirely, enforcing predefined layouts. In another paper [8] performed extensive bench marking on numerous serialization specifications and demonstrated the contrasting variation of output size between *JSON*. While general binary frameworks provide excellent broad-use performance, specialized multimedia contexts frequently demand tailored bit-level specifications. A prime example is the TINY-VG format developed by [9], which strips away the overhead of generic container schema to encode vector graphic primitives into a custom-tailored bit-stream, prioritizing a minimal memory footprint and trivial parsing complexity.

Furthermore, modern hardware optimizations can be leveraged when binary protocols are strategically structured. Traditional variable-length integer fields often trigger frequent CPU branch mispredictions due to unpredictable byte lengths. However, byte-oriented layouts permit the usage of Single Instruction, Multiple Data (*SIMD*) vector lanes. Research into vectorized processing, such as Masked V-Byte decoding by [10] and SIMD-driven posting list evaluations by [11], proves that formatting binary data to fit native CPU word boundaries allows processors to decode multiple data blocks concurrently. This literature demonstrates that by carefully co-designing a file format alongside the physical data structures of modern processors, systems can unlock substantial "*machine efficiency*" gains that general text or compression algorithms cannot achieve. The application of applying modern CPU vectorization to parsing code has been explored in many literature such as [10], [12], [13], [14]. A lot of work aims to improve the performance of JSON without resorting to binary serialization due to its reliance by industry. In cases such as [15], [16] they used *Just-In-Time* compilation to speed up the parsing of JSON. It was stated that they achieved striking results.

This paper aims to evaluate the efficiency of binary formats against the default JSON-based implementation. It will measure the effectiveness of a binary format and what kind of advantage it provides over the default one. It will also try to identify which aspects of a file format makes it ideal for the use-case above. It will measure different metrics across different formats. Rather than focusing on the initial design of a format, this research aims to quantify the "*machine efficiency*" gains achieved when specific binary encoding strategies are applied to the Lottie specification.

2. METHODS AND METHODOLOGY

This study is a Quantitative Experimental Framework designed to measure the impact of a binary serialization on the *Lottie* format. The research is structured as a Comparative Analysis where various file formats serve as the independent variables, and file size (storage efficiency) and parsing overhead serve as the dependent variables. By isolating these variables, the study identifies the "*machine-efficiency*" gains of a specialized binary format over the industry-standard JSON. As a design artifact to test our main hypothesis of a binary serialization being more effective for Lottie data we are using a pre-production implementation of our encoder and decoder to run the experiments on. Our approach is the following.

2.1 Procedure and Methods

Each animation in the dataset was first processed through a reference encoder to generate the corresponding binary bit-strings. This process involved mapping the hierarchical JSON structure into a statically typed binary specification, effectively eliminating the textual overhead and dynamic typing characteristics of JSON.

1. ***Development of the Benchmarking Instrumentation***. A dedicated performance measurement framework was engineered to capture high-resolution metrics. This instrumentation was designed to isolate serialization latency and storage overhead, ensuring that environmental noise was minimized during data collection. The framework will be designed in the language in which the implementation of the

encoder and decoder was built in. This will allow for more granular control of measuring of time and allows for more accurate counting of memory allocation.

2. ***Dataset Curation.*** A corpus of animation documents will be established. The data will be selected in a way that represents a wide variety of structural differences and complexities. This could range from high density of images or an innumerable amount of vector sequences. These vector sequences could be a small part in a hierarchy which describes shapes or a path.

3. ***Comparative Benchmarking.*** Each *Lottie* document will be subjected to a series of benchmarks which follows comparative evaluation. It is important to note that measuring selected metrics on some file formats is not as straightforward and as such will not be included in this study. The raw values of the benchmarks will be pooled together for further analysis.

4. ***Multi-Dimensional Storage Analysis.*** The study quantified the physical footprint of the resulting bit-strings across multiple states. This involved measuring the raw bytes size of the resulting bit string produced by the encoder. The study will focus more on the raw result of the encoder without applying any general compression algorithms such as *G-zip* or *Brotli*. This is due to the reason that we intend to measure a baseline against the current state-of-the-art in *Lottie* serialization without resorting to general compression.

5. ***Critical Discussion and Performance Synthesis.*** The final phase involved a rigorous analysis of the empirical results. This discussion aims to identify the specific architectural characteristics of space-efficient binary specifications and to elucidate the synergistic role that data compression plays in optimizing the transmission and storage efficiency of Lottie assets. This will give light to areas of improvement with the current implementation and opens up possibilities of new architectural changes.

2.2 Measured Metrics

To answer the research question, following metrics were captured from the benchmarks in order to identify the impact of binary serialization.

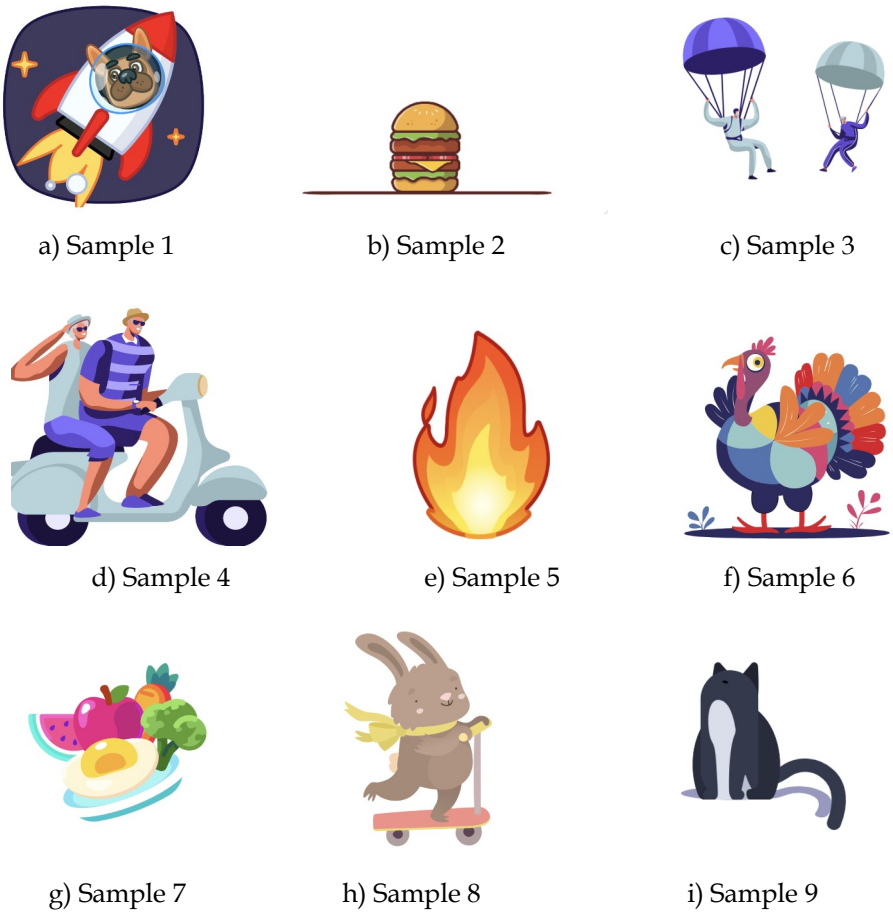
- *Total Size*. Measured number of raw bytes from each of the categories
- *Reduction*. Calculate the proportional decrease from an original size to the new size after serialization. It quantifies how much smaller the binary file is compared to the original JSON file. The reduction can be measured as follows.
  - $R\% = \frac{|S_{orig} - S_{new}|}{S_{orig}} \times 100$
- *Compression Ratio*. It is a quantitative metric used to measure the efficiency of a storage format. It compares the size of the baseline size to the processed size. Gives a conceptual understanding of how many times the data was "*shrunk*". It can be measured as given. $C_{ratio} = S_{base} \div S_{new}$

- *Average Size of Category*. Given $S = \{s_1, s_2, s_3, \dots\}$ are sizes of category C_1 *(Eg: all sizes measured for Tiny-Lottie)*. The average size of S would be expressed as $\sum_{i=1}^{n} S_i \div n$
- *Median*. In the dataset, the file sizes are quite varied. Some files are very small (28 KB), while others are massive *(1,300 KB)*. A single massive file can pull the mean up or down, making the results look better or worse than they actually are for typical files. Given that the number of samples is $N = 9$, median can be calculated as $M = \frac{n+1}{2}$
- *Standard Deviation of Categorical Sizes*. Included to measure how much each files performance differs from the average. It gives a measure of predictability of the target encoder. A low standard deviation means the encoder is able to provide similar level of size of reduction over the dataset relatively. Here $\acute{x}$ is the mean of the category and n is the sample size
  - $s = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \acute{x})^2}{n-1}}$

2.3 Input Data

To ensure a rigorous evaluation of the binary format, the input data was carefully curated to reflect the structural diversity of modern web and mobile animations. Although the dataset ($N = 9$) is limited, it tests the encoders on various challenges of serializing *Lottie*. The selection was primarily based on the below 3 criteria.

- Animations with large numbers of coordinate or Bézier shapes that demonstrate the efficiency of numerical encoding. Given most of the file committed to numerical data this was a criteria to be included.
- Documents with deeply nested hierarchies of layers, shapes, and groups *(e.g., "Bulldog flying on the rocket")*, which evaluate how the specification handles object metadata.
- The dataset spans three orders of magnitude in terms of file size, ranging from a minimal "*Loader Cat*" *(28 KB)* to the "*Colorful Thanksgiving Turkey*" *(1,300 KB)*. This variety allows for the observation of performance scaling across different data volumes.



a) Sample 1 b) Sample 2 c) Sample 3

d) Sample 4 e) Sample 5 f) Sample 6

g) Sample 7 h) Sample 8 i) Sample 9

Figure 1: Full collection of the dataset

Table 1: Characteristic of selected samples

#	Name	Size (KB)	Characteristic
1	Bulldog Rocket	128.45	High metadata-to-path ratio
2	Burger	55.15	Repetitive shape layers
4	Scooter	126.26	Moderate path complexity
5	Fire	337.00	High vertex count; dynamic paths
6	Thanksgiving Turkey	1,300.00	High complexity; large numerical arrays
9	Loader Cat	28.00	Micro-interaction; minimal footprint

2.4 Experimental Setup

The computational experiments were conducted on a workstation powered by an Intel Core i5-10400 processor. This CPU features a base frequency of 2.90 GHz with a maximum turbo frequency of 4.30 GHz, supporting 12 threads. This platform was specifically chosen for its support of the AVX2 instruction set, which is critical for the SIMD-ACCELERATED unpacking procedures discussed in this study. The cache architecture consists of 192 KB L1 cache with a 64B cache-line, a 1.5 MB L2 cache, and a 12 MB shared L3 cache along with 6 GB of DDR4 dual-channel memory operating at 2667 MT/s.

3. RESULTS

Table 2: Comparative Benchmarks of Size in Lottie File Formats (Measured in KB)

#	File Name	JSON	JSON (Opt.)	Dot-Lottie	Dot-Lottie (Opt.)	Tiny-Lottie
1	Bulldog Rocket	128.45	111.52	12.04	11.38	14.42
2	Burger	55.15	29.38	36.89	18.15	0.90
3	Two Parachutists	92.37	77.94	13.52	13.04	14.28
4	Scooter	126.26	107.13	15.20	14.56	18.48
5	Fire	337.00	249.97	86.73	67.08	51.96
6	Turkey	1,300.00	1,220.00	81.60	77.90	172.94
7	Loader Cat	28.00	22.91	3.94	3.52	2.65
8	Rabbit Kick Scooter	73.00	60.15	12.88	12.00	11.42
9	Donut	94.00	77.52	12.96	12.18	12.49

Table 3: Summary of Statistical Metrics Across Formats

Metric	JSON	JSON (Opt.)	Dot-Lottie	Dot-Lottie (Opt.)	Tiny-Lottie
Standard Deviation	404.33	381.86	31.62	27.03	54.40
Mean	248.24	217.39	30.64	25.53	33.28
Median	94.00	77.94	13.52	13.04	14.28
Avg. Deviation	253.44	230.04	25.18	20.87	35.18

Table 4: Recorded Encoding Time Measured in ms

#	File Name	Dot-Lottie	Tiny-Lottie
1	Bulldog Rocket	88.19	1.95
2	Burger	92.92	0.26
3	Two Parachutists	68.69	1.58
4	Scooter	87.47	2.240
5	Fire	97.81	7.22
6	Turkey	123.85	40.56
7	Loader Cat	67.57	1.61
8	Rabbit Kick Scooter	68.09	1.43
9	Donut	67.20	0.47

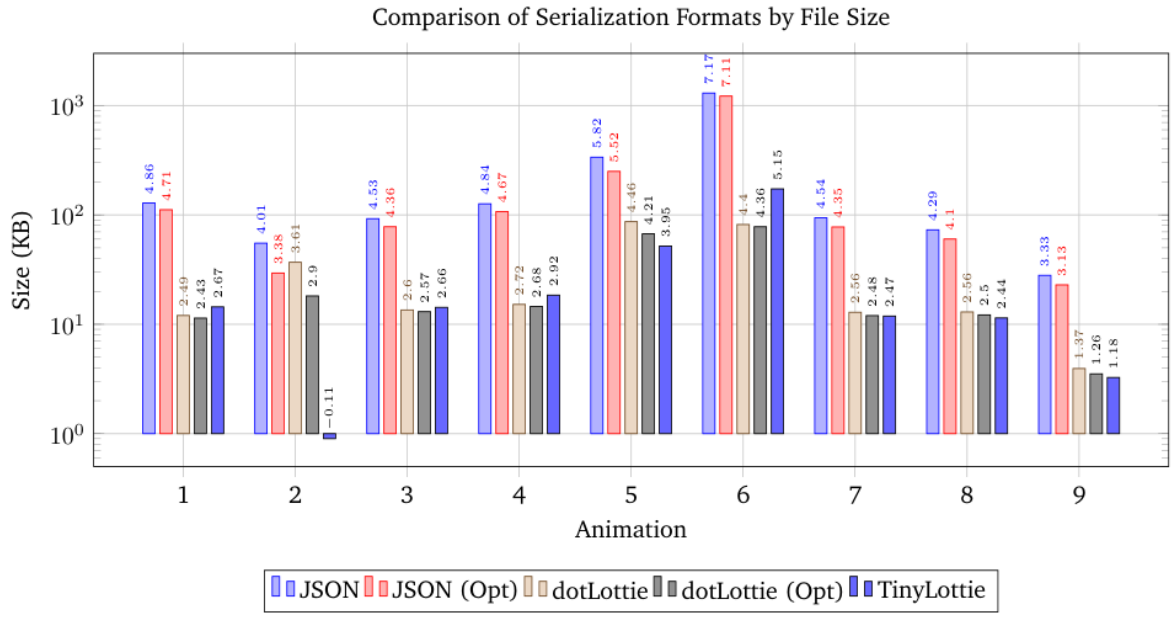


Figure 2: Benchmark results comparing the storage efficiency of various Lottie serialization specifications

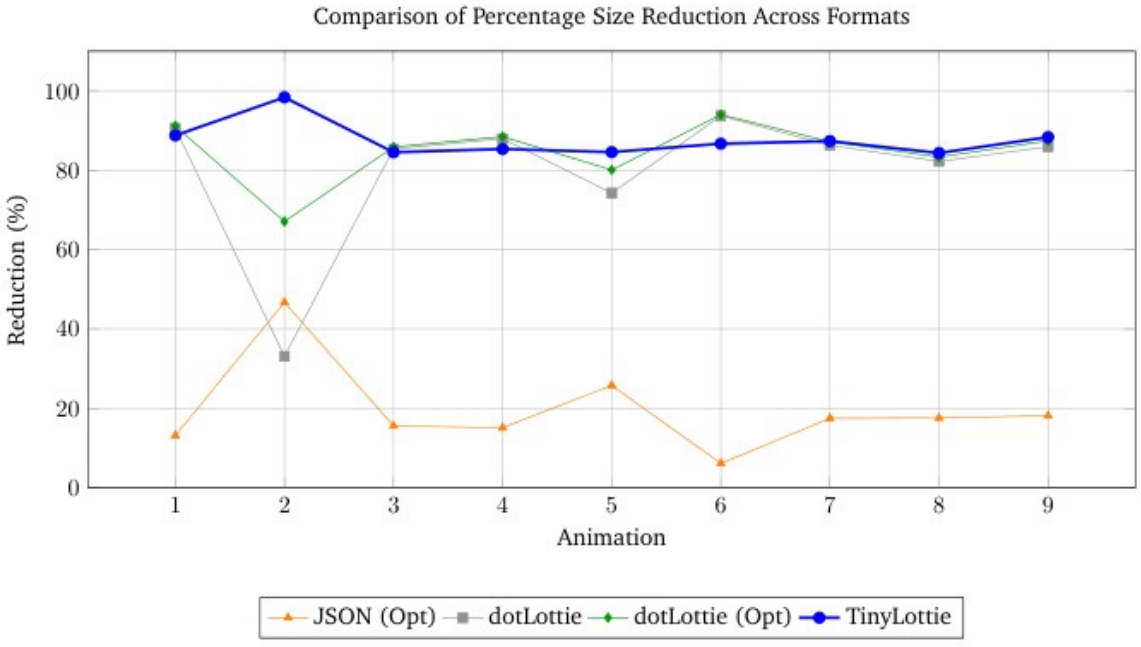


Figure 3: Trend analysis of percentage reduction relative to the original Lottie JSON.

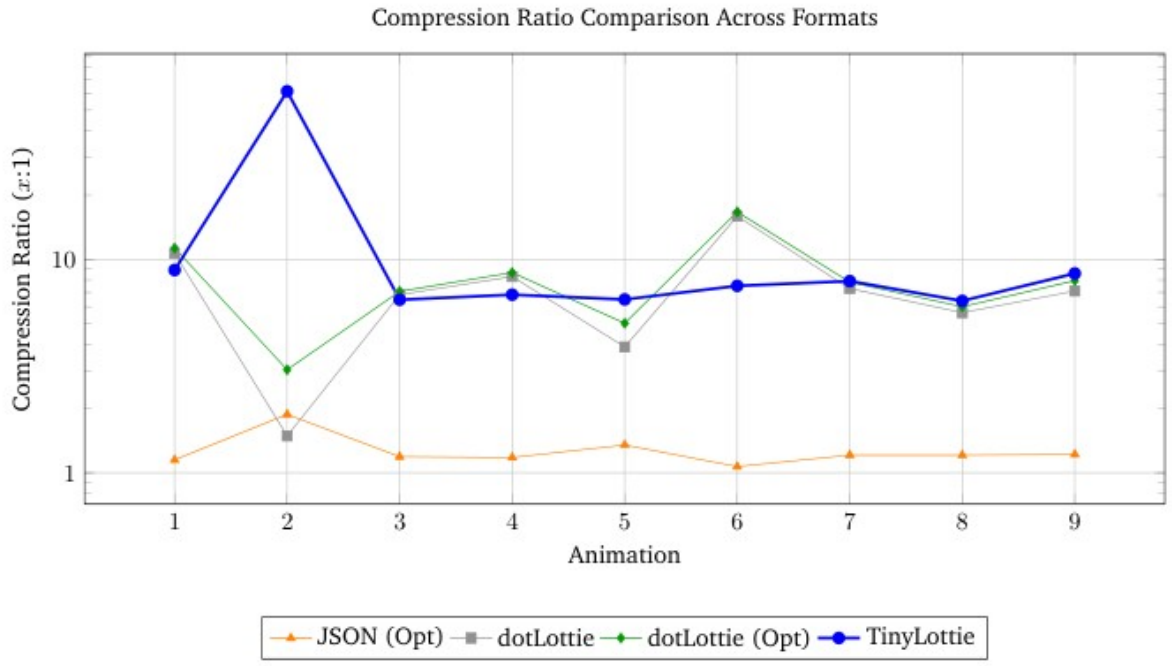


Figure 4: Comparison of compression ratios relative to the Lottie JSON baseline.

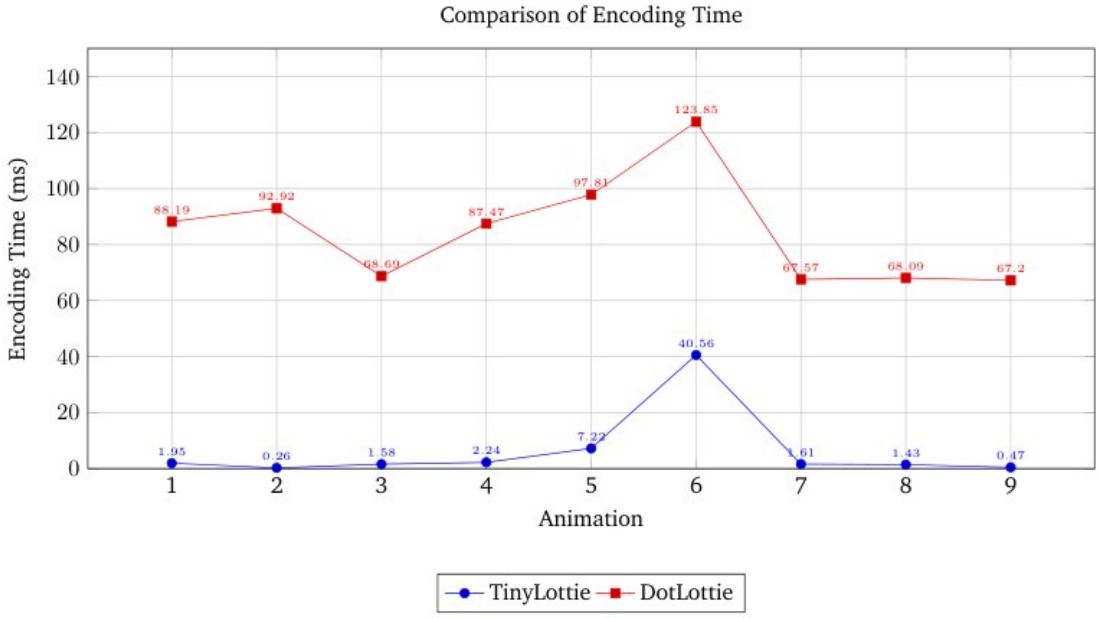


Figure 5: Encoding time comparison between Dot-Lottie and Tiny-Lottie

4. DISCUSSION

The empirical results obtained from bench marking of the dataset in Section 2.3 provide substantial into how binary serialization can influence both storage efficiency and computational overhead. The findings compiled in Section 3 suggest that moving away from textual formats like JSON is not merely a matter of reducing file size, but a fundamental shift in how hardware interacts with animation data. Instead of introducing parser complexity trying to parse complex textual and hierarchical relationships, using a stream interface to dump sequential blocks of data is much more friendly for the CPU.

4.1 Analysis of the Experimental Results

The baseline Lottie JSON format consistently represents the largest footprint, with a mean size of 248.24 KB across the dataset. This inefficiency is primarily due to the textual representation of numerical data and the repetitive inclusion of key-value pair metadata for every layer and frame. While JSON Optimization (minified and key-shortening) offers a slight improvement, reducing the mean to 217.39 KB, it fails to address the underlying scaling issues of the JSON specification, as the data remains a character-based stream.

The data in Table 2 demonstrates that binary formats, specifically TINY-LOTTIE, consistently outperform standard JSON and even minified JSON across all complexity levels. A notable observation is the performance of Sample 2 (Burger), where TINY-LOTTIE achieved a 98.37 % reduction in size. This can be attributed to the format's ability to collapse repetitive shape layers into fixed-width binary structures, avoiding the character-escaping and field-naming overhead inherent in JSON. It is also very likely that such a drastic change is was observed due to the fact, that sample contains little to no image assets. At the moment of writing, image assets are simply copied from its source and dumped into the bit stream after the length. Individual bytes of the image are encoded as *Base64*. It is a binary-to-text encoding scheme that represents binary data in an ASCII string format. While it is incredibly useful for embedding images in HTML, it is generally considered poor practice for high-performance serialization. The most immediate drawback of Base64 is size. Base64 works by taking groups of 3 bytes *(24 bits)* and spreading them across 4 characters *(6 bits each)*. This a recognized flaw of the current encoder implementation.

For samples $S = \{1, 3, 4\}$, the *Dot-Lottie* version outperforms our implementation by a small margin. Although, it still is marginally smaller than both the *JSON* variations. It is not entirely clear the exact reason for this change since there are many factors at play. The transition to *dot-Lottie* represents a major leap in storage efficiency, achieving a mean size of only 30.64 KB, a reduction of over 85 % compared to the *JSON* baseline. This gain is achieved by wrapping the *JSON* data in a zip container and applying the Deflate algorithm, which is highly effective at identifying and compressing the repetitive text patterns characteristic of animation files. The grunt of the work is owed to the *Deflate* algorithm in this case. Being a dictionary based compression algorithm the more entropy in the data there is, the greater the compressibility of the data. This is a possible reason for the outlier results for $S = 6$. *Dot-Lottie* recorded a size of 81.6 KB, but our implementation produced a bit string of 172.94 KB. This is about 93.72 % and 86.70 % in terms of reduction for both categories. As the source file is so large, the deflate algorithm can work better.

One of the goals we are trying to achieve with our encoder/decoder is to refine the encoding and decoding time. While DOT-LOTTIE provides competitive compression, its reliance on the Deflate algorithm introduces a computational bottleneck. The sub-millisecond unpacking times achieved by TINY-LOTTIE for smaller assets $S=\{2,9\}$ indicate that eliminating lexical analysis allows the CPU to transition almost immediately from data retrieval to frame rendering. Figure 5 shows that DOT-LOTTIE has very consistent running times except for *Sample 6*. This was expected due to its size being larger than all the other samples. The same spike was observed for *Tiny-Lottie* with *Sample 6*. *Dot-Lottie* has almost a flat curve with sub-millisecond encoding on almost all the samples. If we take out *Sample 6*, the standard deviation is very low. It is about $S_{dev}=2.18$. This indicates that unless the source file is very large the encoding time almost insignificant.

4.2 Comparison With Literature

The empirical findings of this study reinforce, extend, and in some aspects challenge the established literature regarding text-based serialization bottlenecks and alternative data exchange protocols. The dramatic storage reductions observed when transitioning from Lottie JSON to a specialized binary representation validate the foundational arguments of [3], [4]. Both studies concluded that migrating away from character-based encoding like JSON yields immediate dividends in payload efficiency and transmission speeds. However, while prior literature largely focused on general network communication packets, our results demonstrate that these efficiency gains scale exponentially when applied to structurally dense, high-frequency multimedia files containing large arrays of numerical shape descriptions.

The performance profile of DOT-LOTTIE 2.0 documented in Table 2 aligns closely with the practical paradigms outlined in the [6] specification, proving that general-purpose compression layers (G-zip/Deflate) can successfully mask the baseline textual bloat of JSON assets. Nevertheless, a critical divergence emerges when considering computational unpacking throughput. While general-purpose wrappers hide data overhead on disk, they introduce a sequential, single-threaded processing tax that cannot escape the dynamic execution limits characterized by [15] in their evaluation of loose runtime object parsing frameworks.

By contrast, the architecture of our experimental binary protocol achieves a design sweet spot that bridges the optimization methodologies found across specialized graphics and modern hardware acceleration literature. By eliminating textual variable-length constraints, the layout mirrors the design philosophy of TINY-VG [9] by stripping away generic structural containers in favor of primitive vector streams. More importantly, because numerical coordinate data is encoded in its native, fixed-width binary alignment rather than wrapped in compression tokens, the decoder successfully exploits the vectorized parallel processing strategies detailed by [10], [11]. The sub-millisecond parsing latency accomplished by the binary implementation proves that specialized structural formatting rather than aggressive general compression is the definitive path to achieving predictable "*machine efficiency*" on modern, multi-core CPU architectures.

4.3 Limitations of the Study

(i) With a dataset of only $N = 9$, the results does not capture the properties of most Lottie files. While the samples were curated for structural diversity, a larger corpus would provide a more robust statistical baseline. Despite the fact that sample size was small the results (Section 3) show significant improvements over the status quo. *(ii)* The experiments were conducted on a single hardware configuration (Intel Core i5-10400). While this provides a controlled environment for comparative bench-marking, it does not account for architectural differences found in ARM-based mobile processors or thermal throttling scenarios common in mobile devices. Running on different architectures will help to identify the performance ratios between architectures. *(iii)* The TINY-LOTTIE encoder used is a pre-production design artifact. It lacks support for certain edge-case features of the full Lottie specification (such as complex expressions or certain blending modes), which may slightly increase the file size once fully implemented. *(iv)* Moreover, it was undo-able to perform comparative benchmarks on the decoder since the current implementation is fully on par with any *Lottie* specification. Therefore, it would not produce fairer results.

5. CONCLUSION

This paper has evaluated the impact of novel binary serialization on the delivery and processing of *Lottie* animations files. This study exposed a critical reality: while current industry-standard optimizations like DOT-LOTTIE succeed in minimizing transmission payloads across the network, they do so by trading away valuable client-side computational efficiency. A simple binary serialization format is able to compete without any general compression and has more room for improvement. The empirical results from our bench-marking framework prove that an alternative approach is both viable and highly effective. By transitioning away from textual metadata boundaries in favor of a specialized, schema-driven binary layout, our experimental format TINY-LOTTIE consistently achieved structural reduction ratios exceeding $80.3\times$ across a structurally diverse dataset. This about $7.46\times$ size decrease from the original JSON.

While bounded by the constraints of an undergraduate-level scope, a limited dataset ($N = 9$), and a singular testing environment, this investigation serves as a foundational proof of concept. It demonstrates that maximizing machine efficiency is essential for supporting immediate user experiences in performance-critical or low-bandwidth environments. Future work will focus on scaling the encoder to accommodate edge-case elements of the full Lottie specification and assessing its decoding throughput across heterogeneous mobile processor topologies. It will also prioritize on transforming scalar algorithms used in the implementation to more SIMD friendly algorithms. Ultimately, this research shifts the architectural paradigm of asset delivery from simple storage management to comprehensive runtime efficiency, paving the way for next-generation vector animation frameworks.

Conflicts of Interest: The authors declare no conflict of interest.

Data Availability Statement: The data supporting the findings of this study are available from the author upon reasonable request.

6. REFERENCE

[1] Google, "Protocol Buffers Version 3 Language Specification," Google, 2020.

[2] W. van Oortmerssen, "Flatbuffers," Google, 2014.

[3] C. Puspo Wibowo, "(PDF) Evaluation of Protocol Buffers as Data Serialization Format for Microblogging Communication," in *ResearchGate*, ICID, 2011.

[4] D. P. Proos and N. Carlsson, "Performance Comparison of Messaging Protocols and Serialization Formats for Digital Twins in IoV," *2020 IFIP Netw. Conf. Netw.*, Jun. 2020.

[5] J. C. Hamerski, A. R. P. Domingues, F. G. Moraes, and A. Amory, "Evaluating Serialization for a Publish-Subscribe Based Middleware for MPSoCs," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec. 2018, pp. 773–776. doi: 10.1109/ICECS.2018.8618003.

[6] LottieFiles, "dotLottie 2.0," LottieFiles, 2025.

[7] J. C. Viotti and M. Kinderkhedia, "A Survey of JSON-compatible Binary Serialization Specifications," *ArXiv*, Jan. 2022, Accessed: Oct. 30, 2025.

[8] J. C. Viotti and M. Kinderkhedia, "A Benchmark of JSON-compatible Binary Serialization Specifications," Jan. 09, 2022, *arXiv*: arXiv:2201.03051. doi: 10.48550/arXiv.2201.03051.

[9] F. Queißner, "TinyVG," 2020.

[10] J. Plaisance, N. Kurz, and D. Lemire, "Vectorized VByte Decoding," Jan. 14, 2017, *arXiv*: arXiv:1503.07387. doi: 10.48550/arXiv.1503.07387.

[11] A. A. Stepanov, A. R. Gangolli, D. E. Rose, R. J. Ernst, and P. S. Oberoi, "SIMD-based decoding of posting lists," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, Glasgow Scotland, UK: ACM, Oct. 2011, pp. 317–326. doi: 10.1145/2063576.2063627.

[12] G. Langdale and D. Lemire, "Parsing Gigabytes of JSON per Second," *VLDB J.*, vol. 28, no. 6, pp. 941–960, Dec. 2019, doi: 10.1007/s00778-019-00578-5.

[13] J. Keiser and D. Lemire, "Validating UTF-8 In Less Than One Instruction Per Byte," *Softw. Pract. Exp.*, vol. 51, no. 5, pp. 950–964, May 2021, doi: 10.1002/spe.2920.

[14] D. Lemire and L. Boytsov, "Decoding billions of integers per second through vectorization," *Softw. Pract. Exp.*, vol. 45, no. 1, pp. 1–29, Jan. 2015, doi: 10.1002/spe.2203.

[15] D. Bonetta and M. Brantner, "FAD.js: fast JSON data access using JIT-based speculative optimizations," *Proc VLDB Endow*, vol. 10, no. 12, pp. 1778–1789, Aug. 2017, doi: 10.14778/3137765.3137782.

[16] J. C. Viotti and M. J. Mior, "Blaze: Compiling JSON Schema for 10x Faster Validation," Mar. 11, 2025, *arXiv*: arXiv:2503.02770. doi: 10.48550/arXiv.2503.02770.