University of Aveiro

Information and Coding

Lab work nº 3

Henrique Cruz 103442
Diogo Borges 102954
Piotr Bartczak 130327

December 19, 2025

Contents

1 Introduction

The objective of this work was to compress the file `model.safetensors`, which contains the parameters of a Large Language Model (LLM). The file size is approximately 1 GB. The goal was to maximize the compression ratio while minimizing computation time and memory usage.

2 Analysis of the File Structure

The file `model.safetensors` stores neural network weights in the **BF16 (Brain Float 16)** format. Each number occupies 2 bytes (16 bits):

- **Most Significant Byte (MSB)**: Contains the sign and the exponent. In neural networks, weights often share similar scales (exponents), making this byte highly repetitive and compressible.
- **Least Significant Byte (LSB)**: Contains the mantissa (precision). This part exhibits high entropy and behaves like random noise, making it difficult to compress.

Standard compressors (like Gzip or Zstd) process data sequentially. Because the MSB and LSB are interleaved, the high-entropy LSBs disrupt the patterns of the MSBs, limiting the effectiveness of dictionary-based compression.

3 Proposed Solution: Byte-Plane Splitting

To address the structural characteristics of BF16, we implemented a **Byte-Plane Splitting** strategy. The process involves:

1. **Splitting**: The data stream is separated into two distinct streams: one containing all the even bytes (MSBs) and another containing all the odd bytes (LSBs).
2. **Concatenation**: The two streams are concatenated, grouping the highly compressible MSBs together.
3. **Compression**: The transformed data is compressed using **Zstandard (Zstd)**, a modern algorithm known for its high speed and good compression ratios.

This transformation exposes long sequences of similar bytes (the exponents) to the compressor, significantly improving the compression ratio without loss of information (lossless).

4 Experimental Results

We benchmarked our solution against standard compression algorithms: Gzip, Bzip2, LZMA (XZ), and standard Zstd. The tests were performed on a 100 MB chunk of the file.

4.1 Compression Ratio

Figure 1 shows the compression ratio obtained by each method.

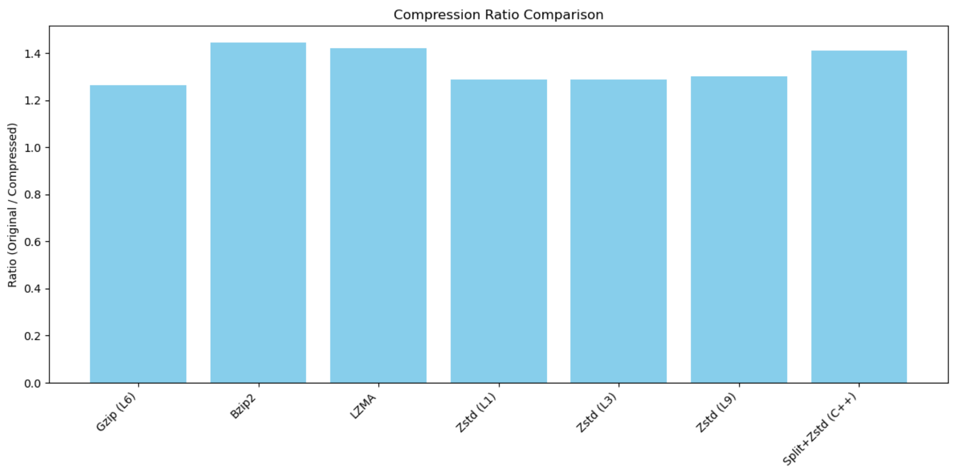

Figure 1: Compression Ratio Comparison (Higher is better)

Our solution (`Split+Zstd`) achieved a compression ratio of approximately **1.41**, outperforming standard Zstd (1.29) and Gzip (1.26), and matching the performance of LZMA (1.42) and Bzip2 (1.44).

4.2 Performance (Time vs. Ratio)

Figure 2 illustrates the trade-off between compression time and ratio.

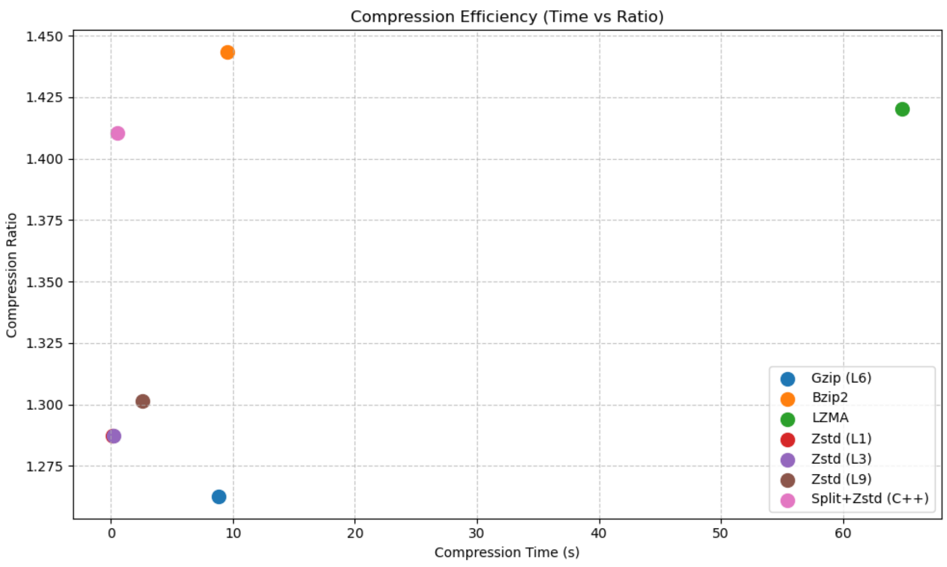

Figure 2: Efficiency Comparison: Time vs. Ratio

Algorithm	**Ratio**	**Comp. Time (s)**	**Decomp. Time (s)**
Gzip (L6)	1.26	8.85	0.64
Bzip2	1.44	9.58	4.92
LZMA	1.42	64.83	2.98
Zstd (L3)	1.29	0.21	0.13
Split+Zstd (C++)	**1.41**	**0.54**	**0.25**

Table 1: Detailed Benchmark Results

As shown in Table 1, `Split+Zstd (C++)` offers the best balance. It is **16x faster** than Gzip and **17x faster** than Bzip2, while providing a superior compression ratio. Compared to standard Zstd, it improves the ratio by $\approx 10\%$ with only a slight increase in processing time (due to the splitting overhead).

5 Conclusion

The **Byte-Plane Splitting** strategy combined with **Zstd** proved to be the optimal solution for compressing the `model.safetensors` file. By exploiting the internal structure of the BF16 format, we achieved a compression ratio of **1.41** with extremely fast compression and decompression times (< 0.6 seconds for 100 MB). This approach is significantly more efficient than traditional general-purpose compressors for this specific type of data.