

SMART METER DATA COMPRESSION: A SOLUTION FOR REDUCING AND OPTIMISING ENERGY METER RESOURCES IN NBIOT NETWORKS <https://doi.org/10.56238/sevened2025.011-017>

Gleison Guardia¹, Rogério Guerra Diogenes², Iury Gonçalves França³, Lucas Noé dos Santos Santana⁴, Jamilly Cristina de Sousa⁵, Brunna Conceição de Paulo⁶, Antônio Ébano Rafael Machado de Oliveira⁷ and Alberto Alexandre Moura de Albuquerque⁸

ABSTRACT

This study proposes a strategy to optimize data compression in embedded devices based on the STM32WBA microcontroller. The main objective was to reduce the volume of data transmitted in NB-IoT networks, with the aim of reducing the volume of this data without compromising the integrity of the information, while at the same time while respecting the memory and computing capacity constraints of these systems. The research used the

¹ Mathematician and Data Scientist (Master's Degree)

Evolution Institute of Science and Technology/Federal Institute of Rondônia – IFRO

E-mail: gleison.guardia@ifro.edu.br

ORCID: orcid.org/0000-0003-1402-0777

LATTES: lattes.cnpq.br/3081488341816997

² Telecommunications Engineer (Master's Degree)

Center for Studies and Research in the North and Northeast – NEPEN

E-mail: rogerio.diogenes@nepen.org.br

ORCID: orcid.org/0009-0007-5461-7937

LATTES: lattes.cnpq.br/3782482898648331

³ Systems Analyst and Developer (in progress)

Evolution Institute of Science and Technology

E-mail: iurygfranca@gmail.com

ORCID: orcid.org/0009-0007-8535-0139

LATTES: lattes.cnpq.br/5608507491741074

⁴ Mechatronics Engineering (in progress)

Center for Studies and Research in the North and Northeast – NEPEN

E-mail: lucas.santana@nepen.org.br

ORCID: orcid.org/0009-0009-6236-1753

LATTES: lattes.cnpq.br/6086974571374063

⁵ Mechanical Production Engineer (Graduate)

Evolution Institute of Science and Technology

E-mail: jamillycristina90@gmail.com

ORCID: orcid.org/0009-0001-8820-3165

LATTES: lattes.cnpq.br/6374960584977744

⁶ Computer Engineering (Studying)

Evolution Institute of Science and Technology

E-mail: brunnacdepaulo@gmail.com

ORCID: orcid.org/0009-0003-6448-1309

LATTES: lattes.cnpq.br/5168826144828472

⁷ Mechatronics Engineering (in progress)

Evolution Institute of Science and Technology

E-mail: eebanorafael@gmail.com

ORCID: orcid.org/0009-0006-7576-2722

LATTES: lattes.cnpq.br/9847201110211140

⁸ Electrical Engineer (Master's Degree)

Evolution Institute of Science and Technology

alberto.alexandre@evolucaoinstituio.org.br

ORCID: orcid.org/0009-0003-1742-8652

LATTES: lattes.cnpq.br/1776411644533237



HEATSHRINK library, based on the LZSS algorithm, adjusting parameters such as the size of the compression window (`window_size`) and the size of the lookahead (`lookahead_sz2`) integrated with the BZ2 algorithm based on the Burrows-Wheeler Transform (BWT) technique, to achieve a balance between compression rate, processing time and memory consumption. The experiments revealed that intermediate configurations, such as HSWB06_HSLB04 and HSWB06_HSLB05, achieved compression rates of up to 46.93% for sets of 96 samples. Combined with a 21.15% reduction in the DLMS header, these configurations resulted in a total compression gain of 68.08%. These parameters also had moderate processing times, in the range of 525 ms, and a memory consumption of 200 bytes, making them suitable for devices with limited resources. On the other hand, more extreme configurations, such as HSWB10_HSLB09, showed inferior performance, with compression rates below 40% and processing times of over 7,900 ms, making them unfeasible for systems that require low latency. A strategic approach to allocating microcontroller resources was also implemented. Original data was stored in FLASH memory, compressed data in RAM and temporary buffers were managed by STACK, ensuring operational stability even under high load conditions. The viability of these configurations was confirmed through memory analyses carried out with the Build Analyzer tool integrated into STM32CubeIDE, which showed minimal impact on the device's operations. The results obtained highlight the potential of the HEATSHRINK library as an efficient solution for data compression in embedded devices, provided it is properly adjusted to the specifics of the application. In addition, the study opens up avenues for future advances, such as the integration of hybrid compression techniques with machine learning, validations in real environments and adaptations for intermittent transmission systems, such as NB-IoT and LoRaWAN networks. These improvements have the potential to significantly expand compression applications in diverse sectors, including energy, health and transport, promoting greater energy efficiency, robustness and reliability.

Keywords: Data compression. Embedded devices. IoT networks. STM32WBA52CE.



INTRODUCTION

This study was conducted by the Evolução Institute of Science and Technology, in collaboration with the North and Northeast Studies and Research Centre (NEPEN) and the Federal Institute of Education, Science and Technology of Rondônia - IFRO, with the aim of investigating and implementing more effective solutions for data compression in embedded device networks.

The research focused on the use of the STM32WBA microcontroller, used in a NIC with NB-IoT technology adapted for the WASON aMeter. The main purpose is to minimize the volume of data transmitted on a daily basis without compromising the integrity of the information, promoting less use of the frequency spectrum and savings in data usage, the main parameter used by telephony operators to charge tariffs.

Wasion Group is a leader in advanced metering, smart distribution and energy efficiency management solutions, operating in more than 50 countries. Founded in 2000 and listed on the Hong Kong Stock Exchange since 2005, the company employs more than 6,000 people and invests around 9.4% of its revenue in R&D, standing out for its technological innovation. Its portfolio covers electricity, water and gas meters, telecoms systems, renewable energy and smart distribution, with more than 100 million devices delivered globally. Wasion has factories in strategic locations such as Mexico, Brazil and Hungary, guaranteeing close service and 24-hour technical support. Recognized for its quality and certifications such as CE and KEMA, the company collaborates with renowned brands such as Siemens and ABB, reinforcing its commitment to energy efficiency and sustainable innovation on a global scale.

The project proposes an innovative approach for the significant and lossless reduction of data traffic, using the concept of energy mass memory. This involves the application of pre-processing techniques to reduce the volume of information before compression, the adoption of specific libraries for data compression and decompression, and integration with a web application for transmitting and retrieving the information.

The structure of the paper is organized into six main sections. **Section 2** presents a literature review, exploring relevant studies that underpin the techniques and solutions discussed. **Section 3** describes the characteristics of the embedded hardware used in the meters, their limitations and the specifics of the data collected and transmitted.

Section 4 details the technical developments carried out, including the algorithms and libraries used, as well as the modelling and implementation of the experiments.

Section 5 discusses the results obtained, analyzing the effectiveness of the proposed



solutions. Finally, **Section 6** presents the study's conclusions, highlighting its contributions and suggesting possible directions for future work.

LITERATURE REVIEW

Data compression for embedded devices has been widely investigated in different contexts, with a focus on improving energy efficiency, reducing the consumption of computing resources and optimising the transmission of information. Ketshabetswe et al. (2021) carried out a comparative analysis of compression algorithms aimed at wireless sensor networks (WSN), highlighting ALDC (Adaptive Lossless Data Compression) and distributed compression and data aggregation methods. This study resulted in the development of an algorithm capable of reducing energy consumption by up to 76.8%.

In the field of textual data compression, Kodituwakku and Amarasinghe (2010) evaluated lossless algorithms, identifying the limitations of LZW in large files and the robust performance of the Huffman and Adaptive Huffman methods. In parallel, Sandoval et al. (2020) introduced tensor decomposition to identify patterns and anomalies in power systems, contributing to the efficient operation of these systems. Zhang et al. (2023) and Zhang (2023) presented RSDC (Real-Time Synchrophasor Data Compression), a real-time technique that has increased compression rates and reduced delays in power grids.

Smart meter studies highlight practical approaches. Santos et al. (2023) investigated NB-IoT meters to reduce fraud in urban and rural areas, recording an increase of 134 kWh per unit after installing protections, with 99% efficiency in data transmission. Meffe et al. (2023) implemented a low-cost solution for remote monitoring with LoRaWAN, providing improved coverage in hard-to-reach locations and greater effectiveness in detecting fraud.

Other contributions explored specific solutions for compression in embedded devices. Lee et al. (2016) proposed a lossless method for mobile data tables, reducing file size without jeopardizing accuracy. Moon et al. (2018) analysed lossy compression in spatio-temporal IoT data, examining the trade-offs between data reduction and information integrity. Qin et al. (2020) introduced the CZ-Array algorithm for long data streams on sensors with limited resources, outperforming gzip and bzip2 in compression rates.

The adaptation of classic algorithms for IoT networks was also explored in Júnior and Oyamada (2023) and Júnior et al. (2023) where they applied methods such as LZ77, LZ78 and Huffman, achieving reductions of up to 22% in energy consumption and 70% in data compression.

Finally, Malandrino et al. (2024) contributed approaches to smart cities and deep neural networks. Gomes applied machine learning to sensor compression, improving

energy efficiency by 22% and reducing latency. Malandrino developed PACT (Predictive Adaptive Compression Technique), dynamically adjusting compression in DNNs and significantly reducing energy consumption.

While this research makes progress in optimizing data compression and energy efficiency, it does not fully address memory constraints, a critical aspect for embedded devices. Based on these studies, this work proposes an innovative solution to reduce the size of files transmitted by controller boards, ensuring that memory consumption and storage capacity remain compatible with hardware limitations during data compression and transmission.

INFRASTRUCTURE

ABOUT HARDWARE

The STM32WBA is a 32-bit wireless microcontroller developed by STMicroelectronics, designed to meet the needs of applications that demand Bluetooth® Low Energy (BLE) connectivity and high energy efficiency. Equipped with the Arm® Cortex®-M33 core, operating at up to 100 MHz, the device combines robust computing performance with advanced security features such as TrustZone® and the Memory Protection Unit (MPU). In addition, it supports DSP instructions and incorporates a Floating Point Unit (FPU), features that make it suitable for digital signal processing, see **figure 01**:

Figure 01: Wasion meters: On the left is the complete meter, in the centre is the top of the STM32WBA microcontroller, on the top right is the front of the microcontroller and on the bottom right is the back side microcontroller.



Source: Author's own

In the field of connectivity, the STM32WBA integrates a 2.4 GHz radio transceiver compatible with BLE 5.4 and additional protocols such as Thread, Matter, Zigbee® and proprietary solutions. The device's reception sensitivity is -96 dBm for BLE at 1 Mbps and -97.5 dBm for IEEE 802.15.4 at 250 kbps, while the output power is programmable up to +10 dBm, in 1 dB increments, offering flexibility for various operating conditions.



As far as memory is concerned, the STM32WBA offers up to 1 Mbyte of flash memory with error correction (ECC), 256 Kbytes of which can support 100,000 write cycles. It also has 128 Kbytes of SRAM, 64 Kbytes of which are parity-checked. The version implemented in smart meters, however, has a simplified configuration, with 512 Kbytes of flash memory and 96 Kbytes of RAM, suitable for more restrictive applications.

The microcontroller is designed to operate in low power consumption environments. In Standby mode, it consumes just 160 nA with 16 wake-up pins, and 6.5 μ A in Stop mode with 64 Kbytes of SRAM, making it ideal for battery-powered devices.

In addition to low power consumption, the STM32WBA is equipped with a wide range of peripherals. This includes a 12-bit ADC with a sampling rate of up to 2.5 Msps, low-power comparators, 16- and 32-bit timers, and communication interfaces such as I2C, SPI, USART and SAI. In addition, the device has a capacitive touch controller that supports up to 20 sensors, extending its versatility for touch-sensitive interface applications.

These specifications make the STM32WBA a highly adaptable and reliable solution, ideal for applications ranging from the Internet of Things (IoT) to industrial systems. Its combination of robust connectivity, energy efficiency and support for a variety of protocols positions it as a strategic choice for projects that require high performance and efficient operation in embedded devices.

ABOUT DATA

The study was based on the Wasion three-phase meter, in which data is transmitted via a Network Interface Card (NIC) to a Data Collection Module (MDC). During this process, the data relating to the meter's measurements is sent in the form of a mass memory frame, structured as an array of bytes. This array encapsulates the information related to the measurements taken by the device.

The DTSD341 three-phase energy meter, developed by Wasion, is a solution for residential consumers and small commercial establishments, offering accurate and reliable measurements, see **figure 01**. Certified with class 1.0 accuracy for active energy and class 2.0 for reactive energy, the device supports voltage ranges of 3x120/240 V and 3x127/220 V, with a tolerance of -20% to +20%.

Equipped with an integrated RF module for wireless communication and an optical port option, the DTSD341 facilitates remote reading and parameter configuration, guaranteeing operational efficiency. Its features include support for up to four different tariffs, detailed load profile recording with data storage for 60 days (at 15-minute intervals), and detection of events such as cover opening, current reversal and phase failures. Its

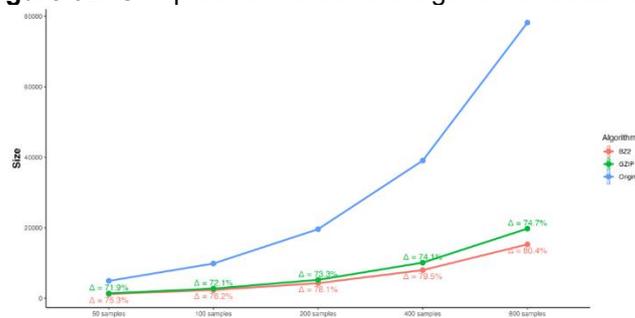
increase the gain obtained in reducing the volume of bytes transmitted, optimizing transmission efficiency and ensuring greater savings in the use of system resources.

DEVELOPMENT

In order to evaluate and improve the performance of this work, an extensive literature search was carried out in renowned sources, including the Data Compression Conference (DCC), the IEEE (Institute of Electrical and Electronics Engineers) database, the CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) journal portal and the proceedings of the 2024 SENDI (Seminário Nacional de Distribuição de Energia Elétrica). The research focused exclusively on topics related to data compression in embedded systems.

Based on the studies identified, algorithms were selected for the initial tests, prioritizing those with the potential to meet the system's needs in terms of efficiency and performance. Based on this selection, a detailed analysis of the compression applied to mass storage was carried out, varying the amount of data processed. The aim was to find the best balance between data volume, compression rate and algorithm performance. The results obtained for each configuration tested are shown in **figure 02**.

Figure 02: Compression rate of the algorithms researched



Source: Author's own

The analysis revealed a reduction in the volume of data transmitted ranging from 75% to 80%, highlighting the relevance of the proposed implementation. It was also observed that an increase in the number of files accumulated results in incremental gains in the compression rate. However, the time interval needed to send the files proved to be a determining factor in this process.

According to the system data, accumulating 100 readings in the mass memory requires a period of approximately 24 hours. As the additional percentage gain in compression does not justify the additional time needed to accumulate more data, it was decided to standardize on a 24-hour interval as the basis for sending. This decision seeks



to balance compression efficiency and processing time, while maintaining the practical viability of the system.

ABOUT THE BZ2 ALGORITHM

The BZ2 compression algorithm is widely recognized for its efficiency in lossless data compression and is a frequent choice on Unix/Linux systems. Developed in the 1990s, it is based on the Burrows-Wheeler Transform (BWT) technique, which reorganizes substrings to facilitate compression. Compared to traditional methods such as ZIP and GZIP, BZ2 has higher compression rates in many cases, making it an effective solution for storing and transmitting data \cite{burrows1994bwt}. Its effectiveness comes from the combination of advanced techniques, such as BWT, Move-to-Front (MTF), Run-Length encoding (RLE) and Huffman encoding, which optimize the compression process Seward (2024).

How BZ2 works is made up of structured stages that maximize compression. Firstly, the Burrows-Wheeler Transform (BWT) rearranges the characters in the text, grouping repetitive patterns and preparing the data for the subsequent phases. Next, Move-to-Front (MTF) converts characters into indices based on their frequency, maximizing symbol repetition Cleary and Witten (1984). Run-Length Encoding (RLE) then compresses repetitive sequences, such as turning 'aaaaa' into '5a'. Finally, Huffman Coding assigns shorter sequences to the most frequent characters, further reducing the size of the data Knuth (1998). Despite these advantages, BZ2 requires more memory and processing power, limiting its application in systems with limited resources.

BZ2 is widely used in practices that require efficient compression. On Unix/Linux systems, it is the standard choice for creating compressed files with a .bz2 extension, especially in backups and logs, due to its ability to reduce the size of large files and facilitate their transfer Python Software Foundation (2024). In addition, BZ2 is ideal for compressing large volumes of data before transmission, reducing costs and optimizing processes. Tools such as tar offer native support for BZ2, integrating it efficiently into workflows.

In this study, BZ2 was assessed for its viability in embedded systems, analyzing the use of ROM and RAM memory, as well as the operating cost. However, the nature of the data to be compressed, stored in byte format, presented significant challenges. Many compression algorithms are optimized to work with strings, hexadecimal values or floating point numbers, while the data in this project required a specific approach. This limitation led to experimentation with different libraries, looking for solutions that met the system's constraints.



After careful analysis, the decision was made to adopt the HEATSHRINK library, available in the <https://github.com/atomicobject/heatshrink> repository. Developed for embedded devices, HEATSHRINK is lightweight and efficient, designed to operate with data formats such as bytes, meeting the specific needs of this project. The choice of HEATSHRINK allowed the challenges posed by the original data to be circumvented and made it possible to implement efficient compression in line with the limitations and demands of the system.

ABOUT HEATSHRINK

The Heatshrink library is a solution designed specifically for data compression and decompression in embedded systems and real-time environments. Its main feature is its low memory consumption, with minimum requirements of just 50 bytes, as well as its ability to process data incrementally, optimizing CPU usage in a controlled and efficient manner. This flexibility makes Heatshrink ideal for devices with limited resources, allowing static or dynamic memory allocations according to the project's needs. The settings can be customized directly in the *heatshrink_config.h* file, providing precise adjustments for different applications.

The library offers two forms of integration. The developer can directly incorporate the encoding and decoding functions into the project or use the stand-alone command-line programmed for independent operations. Operation follows a simple model based on a state machine. The input data is supplied using the sink method, processed and retrieved using the poll method, and finally the input is closed with the finish method. This cycle can be repeated as many times as necessary, allowing incremental manipulation of the data.

With its efficiency and ease of use, Heatshrink stands out as a dynamic tool with a low computational cost, making it a popular choice for embedded systems that require data compression without compromising the device's limited resources.

ABOUT CODE

To begin implementing the tests, we structured our algorithms in such a way as to integrate the coding of the BZ2 algorithm with the libraries provided by Heatshrink. This combined approach aims to leverage the advantages of both methods, optimizing data compression in embedded systems.

The **Table 03** presents the pseudocode that describes the logic used to implement the data compression process. This structure details the fundamental stages of the algorithm, demonstrating how the data is initially processed by BZ2 and then refined using

Heatshrink. This integration was designed to maximize efficiency while respecting the memory and CPU limitations typical of embedded devices.

Table 03: Compression pseudo code

```

01 Use stdio.h
02 Use string.h
03 Use heatshrink_encoder
04 Use data_compress_app_layer.h

05 Define max_compressed_data
06 Create mass_memory_samples
07 Create compressed_data ← 0xff

08 Function compress_data (p_from*, from_size, p_to*, p_to_size*):
09     Initialize p_from*
10     Initialize from_size
11     Initialize p_to*
12     While ( $x_i \leq x_n$ ):
13          $p\_to^* \leftarrow x_i$ 
14     If  $p\_to^* \geq max\_compressed\_data$ 
15         Print(Error)
16         Break
17     Update p_to_size*
18     Return: p_to_size*

19 Function compress_data_test():
20     Initialize compressed_data
21     compressed_data ← compress_data(mass_memory_samples)
22     Print compressed_data

```

Source: Author's own

The process seeks to detail the implementation of the data compression system using the Heatshrink library, with an emphasis on simplicity and efficiency. Initially, fundamental constants are defined, such as the maximum size allowed for the compressed data, as well as basic structures, including the array *mass_memory_samples* to store the memory samples that will be compressed and the array *compressed_data* to contain the compressed results, initially filled with default values.

The main function, *compress_data*, is responsible for performing the compression iteratively. It takes as input the original data, pointers to store the compressed results and a variable to record the final compression size. The flow of this function involves initializing the encoder, processing the original data in blocks, and continuously extracting the compressed data until all the content has been processed. In the event of errors during execution, specific messages are displayed to ensure transparency and facilitate debugging. Once compression is complete, the final size is updated, and the function returns the index of the last data processed.



For validation and testing, the *compress_data_test* function uses the *mass_memory_samples* array as input, displaying the returned index and the compressed data in hexadecimal format directly on the console. This visualization makes it easier to analyze and verify the results obtained.

The implementation also includes auxiliary functions based on the library components, such as *heatshrink_encoder* and *heatshrink_decoder*, which are used for data compression and decompression. In addition, robust error handling mechanisms have been integrated to ensure that the process is reliable and secure, even in adverse scenarios.

With its modular and well-defined structure, this approach is highly suitable for embedded applications that require efficiency and reliability in data compression, aligning performance with simplicity of implementation.

The *decompress_data* **table 04** function is responsible for decompressing the previously processed data, ensuring that the compressed content is restored within the specified limits. It takes as input the compressed data, the size of that data, a pointer to store the decompressed data, the final size of the decompressed data and the maximum size allowed for the output, ensuring that the process remains within the memory restrictions.

Table 04: Decompression pseudo-code

```
01 Use stdio.h
02 Use string.h
03 Use heatshrink_decoder
04 Use datacompressaplayer.h

05 Define max_decompress_size

06 Function decompress_data(p_from*, from_size, p_to_size*, max_decompress_size):
07   Initialize p_from*
08   Initialize from_size
09   Initialize p_to*
10   Initialize p_to_size*
11   While  $x_i \leq x_n$ :
12      $p_{to}^* \leftarrow x_i$ 
13   If  $p_{to}^* \geq \text{max\_decompress\_size}$ 
14     Print(Error)
15     Break.
16   Update p_to*
17   Update p_to_size*
18   Return: 0
```

Source: Author's own

The function starts by configuring the decoder and initializing the control variables, such as the processing rate and the size intended for the decompressed data. To ensure a consistent state, the decoder is restarted before receiving the compressed data, which is gradually fed into smaller chunks. This split-into-fragments approach reduces the risk of



overloading memory, ensuring efficient processing that is compatible with the system's limitations.

During each processing stage, the function carries out rigorous checks to identify possible errors. If a fault is detected, error messages are displayed, facilitating diagnosis and correction. At the same time, the processing index is continuously updated, keeping pace with the progress of data handling. As the flow progresses, the decompressed data is extracted in each iteration until all the compressed content has been completely processed.

At the end of the decompression process, the final size of the decompressed data is recorded in the pointer provided, ensuring that the memory allocated corresponds to the amount of information recovered. Finally, the function returns the value 0, signaling that the operation has been successfully completed and that the original content has been restored without errors or interruptions.

IMPLEMENTATION

The HEATSHRINK library uses a set of configurable parameters that directly impact its efficiency in terms of compression and resource use. As described in the documentation, the *window_size* parameter defines the size of the window, expressed as 2^W bytes, and has a direct influence on both memory consumption and the efficiency of the compression algorithm. Another relevant parameter is *lookahead_sz2*, which specifies the maximum length of repeated patterns, also defined as 2^W bytes, with ideal values between 3 and $window_size - 1$. In addition, the *input_buffer_size* parameter determines the size of the input buffer in the decoder, influencing the amount of data processed at each stage.

In the experiments carried out, the initial setting for the HEATSHRINK_STATIC_WINDO_BITS-(W) parameter was set to 4, gradually increasing to 10. Similarly, the HEATSHRINK_STATIC_LOOKAHEAD_BITS-(L) parameter was adjusted from 3 to $n-1$, taking the maximum *window_size* value into account. Each experimental battery included 30 different combinations of parameters. The test scenario was designed to assess performance in two data collection contexts: the first with 96 measurements, equivalent to 24 hours of readings taken at 15-minute intervals; the second with 144 measurements, corresponding to readings every 10 minutes over the same 24-hour interval. This last scenario was called Collection Time (T_c).

The main objective of the experiments was to identify the configuration that would provide the best balance between the time needed to send the data and the compression efficiency, maximizing the cost-benefit ratio. The mathematical structure used to model the

configurations and analyse the results is formalized in **equation 01**, which describes the methodological approach applied during the tests.

$$T_{ij} = W_i L_j, \quad (1)$$

where:

- $i = \{4, 5, 6, \dots, 15\}$
- $j = \{3, \dots, i - 1\} \quad j \in Z.$
- T represents the Test

EXPERIMENTS

To organize the experiments, a factorial test design was adopted, represented by:

$$T_{30} \times T_{c_2},$$

resulting in 60 different tests, covering all possible combinations of configurations.

The HEATSHRINK library was used to evaluate its effectiveness in compressing mass memory frames from a three-phase meter. Each frame contains 52 bytes, of which 41 bytes are made up of useful data after removing the DLMS header, resulting in an initial reduction of 21.42% in the volume of data transmitted. These frames include critical information such as the UNIX timestamp, active and reactive energy totalizers under different operating conditions, as well as the voltages and currents corresponding to phases A, B and C.

The implementation of the HEATSHRINK library, based on the LZSS algorithm, made it possible to adjust parameters that balance compression efficiency with system resource consumption. The maximum value for the `window_size` parameter was limited to 10, due to the device's memory restrictions, which is equivalent to a buffer of 2,048 bytes. The experiment was designed to test various combinations of parameters, evaluating their impact on memory usage, compression rate and execution time.

The embedded device used has significant memory limitations: 4,096 bytes available for STACK, 2,048 bytes allocated for HEAP and 4,102 bytes free. In FLASH memory, the device has 24,316 bytes. These resources were strategically allocated to ensure system stability during the tests. The original data was stored in FLASH memory (.rodata section), the compressed data was allocated in RAM (.noinit section), and the buffer required for library operation was allocated locally in STACK.



The memory usage analysis, conducted with the help of STM32CubeIDE's Build Analyzer, revealed that the inclusion of the HEATSHRINK library implied an increase in the size of the .text section. Before the library was incorporated, this section occupied 146,204 bytes. After inclusion, usage increased to 148,912 bytes, representing an increase of 2,708 bytes. Of this total, 494 bytes were attributed to the functions created specifically for the tests, while 2,214 bytes corresponded to the library itself. In addition, a variation in memory consumption was identified as the window_size parameter was adjusted: for value 6, there was an increase of 44 bytes; for 7, the increase was 212 bytes; and for values between 8 and 10, the increase varied between 216 and 220 bytes.

STACK's static analysis confirmed that local buffer allocation, when configured properly, did not compromise the integrity of the system's memory. This validation was essential to ensure that the configurations tested were suitable for real devices, guaranteeing the reliability of operations without risk of failure. The results obtained are detailed in **Table 05**, providing a comprehensive overview of the impact of the proposed configurations.

Table 05: Results of the tests carried out, where α represents the size of the original data, β represents the size of the compressed data, Δ represents the compression rate, t represents the time taken for compression and SSA represents Static Stack Analyze.

Configuration	96 Samples				144 Samples				SSA (bytes)
	α_1	β_1	$\Delta_{\alpha_1, \beta_1}$	t	α_2	β_2	$\Delta_{\alpha_2, \beta_2}$	t	
	(bytes)	(bytes)	(%)	(ms)	(Bytes)	(Bytes)	(%)	(ms)	
HSWB04_HSLB03 104	3840	2405	0,3737	439	5760	3674	0,36215	664	
HSWB05_HSLB03 136	3840	2258	0,4120	435	5760	3341	0,41997	661	
HSWB05_HSLB04 136	3840	2311	0,3982	452	5760	3520	0,38889	687	
HSWB06_HSLB03 200	3840	2120	0,4479	541	5760	3179	0,44809	809	
HSWB06_HSLB04 200	3840	2038	0,4693	525	5760	3056	0,46944	786	
HSWB06_HSLB05 200	3840	2045	0,4674	527	5760	3060	0,46875	789	
HSWB07_HSLB03 336	3840	2144	0,4417	840	5760	3212	0,44236	1257	
HSWB07_HSLB04 336	3840	2047	0,4669	821	5760	3070	0,46701	1228	
HSWB07_HSLB05 336	3840	2061	0,4633	821	5760	3084	0,46458	1228	
HSWB07_HSLB06 336	3840	2112	0,4500	830	5760	3160	0,45139	1243	
HSWB08_HSLB03 592	3840	2175	0,4336	1400	5760	3268	0,43264	2103	
HSWB08_HSLB04 592	3840	2064	0,4625	1383	5760	3102	0,46146	2074	
HSWB08_HSLB05 592	3840	2087	0,4565	1392	5760	3127	0,45712	2087	
HSWB08_HSLB06 592	3840	2140	0,4427	1396	5760	3206	0,44340	2092	



HSWB08_HSLB07 592	3840	2223	0,4211	1502	5760	3328	0,42222	2247
HSWB09_HSLB03 1104	3840	2207	0,4253	2532	5760	3320	0,42361	3799
HSWB09_HSLB04 1104	3840	2086	0,4568	2517	5760	3142	0,45451	3773
HSWB09_HSLB05 1104	3840	2115	0,4492	2547	5760	3173	0,44913	3811
HSWB09_HSLB06 1104	3840	2205	0,4258	2764	5760	3309	0,42552	4149
HSWB09_HSLB07 1104	3840	2243	0,4159	2767	5760	3365	0,41580	4153
HSWB09_HSLB08 1104	3840	2281	0,4060	2770	5760	3422	0,40590	4158
HSWB10_HSLB03 2128	3840	2244	0,4156	4712	5760	3380	0,41319	7030
HSWB10_HSLB04 2128	3840	2116	0,4490	4716	5760	3194	0,44549	7008
HSWB10_HSLB05 2128	3840	2178	0,4328	5254	5760	3290	0,42882	7931
HSWB10_HSLB06 2128	3840	2218	0,4224	5254	5760	3350	0,41840	7933
HSWB10_HSLB07 2128	3840	2258	0,4120	5257	5760	3409	0,40816	7936
HSWB10_HSLB08 2128	3840	2298	0,4016	5256	5760	3469	0,39774	7936
HSWB10_HSLB09 2128	3840	2338	0,3911	5265	5760	3528	0,38750	7948
Average 957		2179	0,4325	2318		3276	0,43119	3483

Source: Author's own

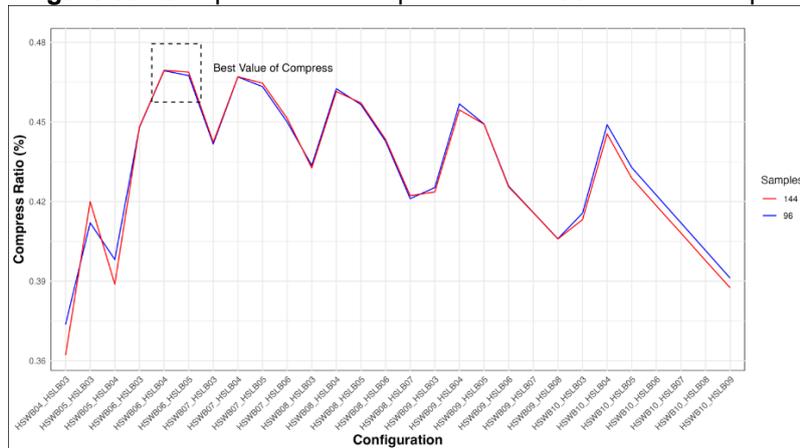
The tests conducted with the HEATSHRINK library clearly demonstrated how the settings of its parameters influence compression efficiency and the system's temporal performance. During the evaluations, varying the `window_size` and `lookahead_sz2` parameters revealed a direct relationship: configurations that adopted higher values for these parameters resulted in higher compression rates, reflecting greater efficiency in data compression. However, this gain was accompanied by a significant increase in the time required for processing.

This behavior highlights the characteristic trade-off between compression efficiency and performance, a central issue in the context of embedded systems. These devices operate under severe resource limitations, such as memory and processing capacity, which requires a careful balance when choosing configurations. Therefore, optimizing the parameters of the HEATSHRINK library must consider not only maximizing compression, but also the need to preserve the overall performance of the system, ensuring that it remains functional and efficient within the constraints imposed by the hardware.

DISCUSSION OF RESULTS

Among the results obtained, the performance of intermediate configurations stood out, such as HSWB0_HSLB04 and HSWB06_HSLB05, which demonstrated a compression rate close to 46.93% when processing 96 samples, with manageable execution times of approximately 525 ms. These findings show that it is possible to achieve a suitable balance between compression efficiency and temporal performance, making these configurations viable for embedded systems that need to optimize resource use without significantly compromising response time. See the **figure 03**.

Figure 03: Compress ratio comparison with 96 and 144 samples

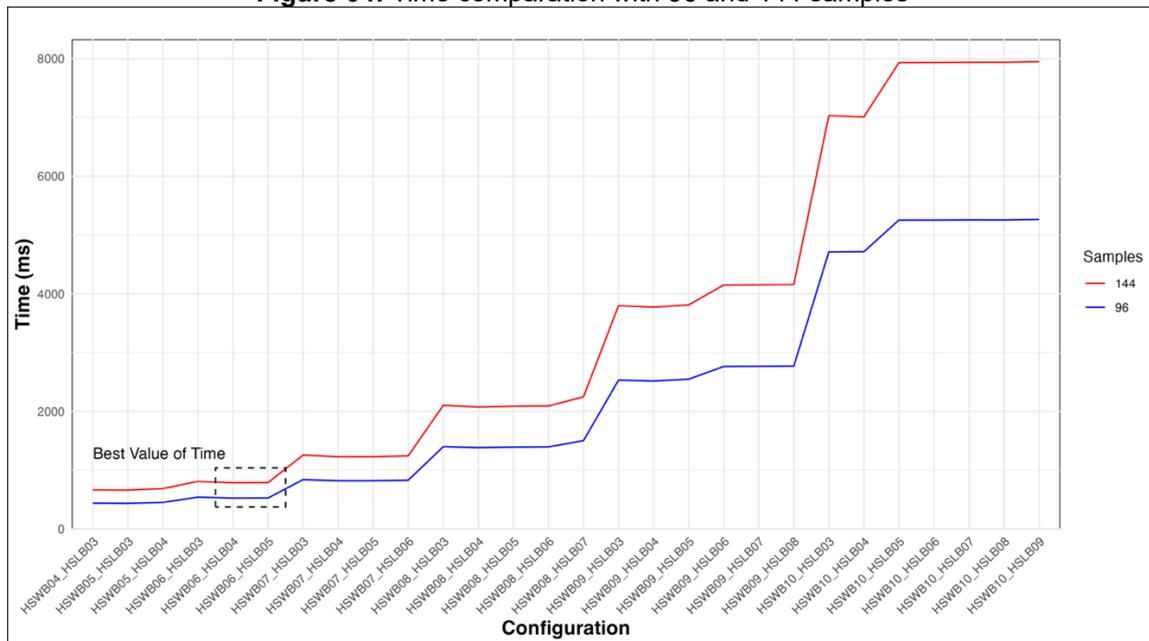


Source: Author's own

In contrast, more extreme configurations, such as HSWB10_HSLB08 and HSWB10_HSLB09, showed inferior performance, with compression rates below 40%. In addition, the processing times for these configurations exceeded 7,900 ms when dealing with 144 samples, a value that makes it unfeasible to use them in applications that require low latency or real-time responses. These results emphasize the importance of carefully balancing the parameters to meet the specific demands of each application, especially in embedded devices where hardware limits impose severe restrictions on the choice of extreme configurations.

The tests also revealed that the number of samples processed directly impacts time performance, while compression rates remain practically unchanged. When comparing scenarios with 96 and 144 samples, a proportional increase in execution time was observed as a function of the volume of data processed, see **figure 04**. These results indicate that the HEATSHRINK library is scalable, allowing it to be used with different volumes of data without compromising compression efficiency. However, in our application and use case, time is not a limiting factor, as it could take up to minutes since we will only be compressing once a day.

Figure 04: Time comparison with 96 and 144 samples

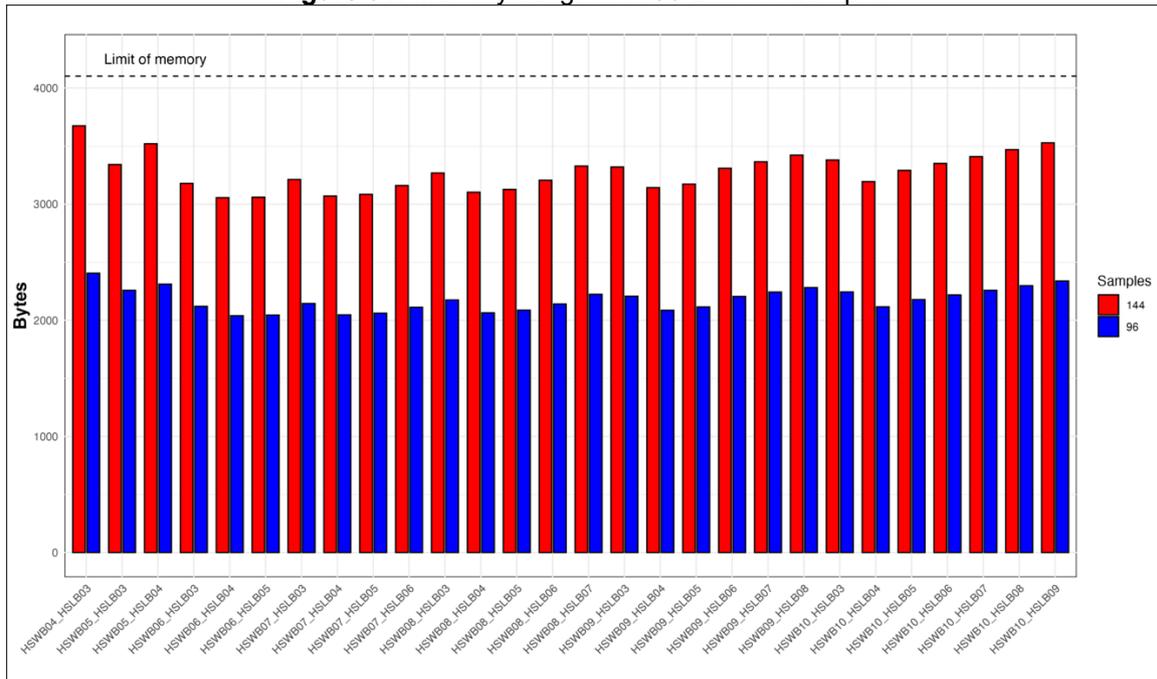


Source: Author's own

However, the impact on execution time highlights the need for precise adjustments to configurations, especially in applications that demand reduced response times. This analysis reinforces that while HEATSHRINK's scalability is an advantage, its application in embedded systems must be carefully designed to ensure that performance meets specific operational requirements while maintaining system stability and efficiency.

Analyzing the impact on memory usage revealed significant results, especially in more robust configurations such as HSWB10_HSLB09, which demanded up to 2,128 bytes of memory, see **figure 05**. This consumption represents a considerable challenge for embedded devices, which are often characterized by limited resources. These findings emphasize the importance of making careful parameter adjustments, considering not only compression efficiency and execution time, but also memory consumption, a critical factor in ensuring the stability and functionality of embedded systems in restricted operating scenarios.

Figure 04: Memory usage with 96 and 144 samples



Source: Author's own

Based on the results obtained, configuration recommendations should be adapted to the specific context of the application. For devices facing strict time and resource constraints, intermediate configurations such as HSWB06_HSLB04 are more appropriate. These configurations offer an advantageous balance, with efficient compression rates and manageable runtimes. In this case, compression proved to be highly effective: by initially reducing the header by 21.15% and applying an additional compression rate of 46.93%, a final reduction of 68.08% was achieved, leaving the original file with only 31.92% of its initial size.

In applications where maximum compression efficiency is a priority, configurations with higher `window_size` and `lookahead` values can be explored, as long as memory consumption remains within the acceptable limits of the hardware. In this way, it is possible to optimize the performance of the HEATSHRINK library to meet both the compression demands and the restrictions of each operating environment, guaranteeing the viability and efficiency of the application in embedded systems.

The results obtained emphasize the importance of a detailed and careful analysis when defining the parameters of the HEATSHRINK library for embedded systems. Proper adjustment of these parameters is fundamental to achieving the ideal balance between compression efficiency, processing time and memory consumption. This approach ensures that applications meet their specific demands without compromising technical feasibility or operational efficiency, especially on devices with severe resource constraints.



The findings also highlight the need for a balanced parameterization of the library. In scenarios where the priority is compression efficiency, higher values for `window_size` and `lookahead_sz2` proved advantageous, providing higher compression rates, even with the increase in memory consumption and processing time. On the other hand, in applications that require low latency, such as real-time systems, intermediate configurations of these parameters were more suitable. These configurations achieved satisfactory compression rates, with shorter execution times and less impact on RAM usage, making them ideal for contexts where fast performance is essential.

The experiment showed that applying HEATSHRINK to embedded systems requires a thorough understanding of the specific needs of the application, as well as the limitations imposed by the hardware. The strategy of allocating the original data in FLASH memory and the compressed data in RAM, combined with strict control over the library's parameters, made it possible to explore its efficiency in different scenarios. This approach not only optimized data storage and transmission, but also guaranteed system stability and performance, even in resource-limited environments.

These results provide a basis for implementing compression solutions in embedded devices. By balancing the configuration of parameters with the specific demands of each application, it is possible to maximize the benefits of compression, guaranteeing the functionality and efficiency of the system without compromising its stability. These findings can serve as a valuable reference for the development and optimization of embedded systems that require robust and scalable data compression solutions.

CONCLUSION

This study investigated the application of data compression techniques in embedded systems, focusing on the use of the HEATSHRINK library to optimize the performance of devices based on the STM32WBA microcontroller. The main objective was to develop an efficient solution that met the resource constraints of this hardware, reducing the volume of data transmitted without compromising the integrity of the information or the stability of the system.

The tests carried out explored different HEATSHRINK parameter settings, with adjustments to the `window_size` and `lookahead_sz2` values to balance compression rate, processing time and memory consumption. Intermediate configurations, such as `HSWB06_HSLB04` and `HSWB06_HSLB05`, achieved a compression rate of approximately 46.93% when processing 96 samples, with execution times of around 525 ms and a controlled impact on memory usage, limited to 200 bytes of RAM. Considering the



additional reduction provided by removing the header, the total efficiency reached 68.08%. These results demonstrate that such configurations are suitable for embedded devices that require a combination of good compression efficiency and manageable execution times.

In contrast, more extreme configurations, such as HSWB10_HSLB08 and HSWB10_HSLB09, showed compression rates of less than 40%. In addition, processing times exceeded 7,900 ms for 144 samples, while memory consumption reached 2,128 bytes. These results emphasize the importance of careful analysis when choosing parameters, particularly in systems with severe memory and latency restrictions.

The analysis also revealed that the volume of samples processed directly impacts temporal performance, even if compression rates remain stable. This behavior indicates that the HEATSHRINK library is scalable but requires precise adjustments to meet the specific demands of each application. For scenarios requiring low latency, intermediate configurations proved to be the ideal choice, offering an efficient balance between compression and performance.

The technical feasibility of the experiment was ensured by strategically allocating the STM32WBA's resources. With RAM memory limited to 4,096 bytes, the original data was stored in FLASH memory, while the compressed data was allocated to RAM. The buffer required for library operation was managed locally in the STACK, ensuring that the system remained stable and free of memory overflows during the tests. The analysis carried out with the STM32CubeIDE Build Analyzer confirmed that these allocation strategies made it possible to integrate the HEATSHRINK library effectively, without compromising the device's overall operations. These results provide a basis for applying data compression to embedded systems, balancing efficiency and stability in different operating scenarios.

The results of this study confirm that data compression, in addition to significantly reducing the volume transmitted, represents an effective solution for extending the life of battery-powered devices, reducing energy consumption and optimizing bandwidth in IoT networks. Considering the specific limitations of the STM32WBA, this work has proposed optimized configurations that maximize the microcontroller's resources without compromising its storage capacity or RAM during the data compression and transmission processes.

As its main contribution, the study positions the HEATSHRINK library, combined with compression algorithms, as a viable solution for compression in embedded devices, provided that it is configured precisely and appropriately to meet the hardware constraints and specific demands of each application. The results presented provide a basis for the development of future implementations, while also suggesting the incorporation of hybrid



techniques and the use of machine learning to further improve the performance and efficiency of devices with limited resources.

The advances made in this study offer a significant contribution to Wasion, strengthening its position as a leader in smart metering and energy efficiency solutions. The detailed analysis of the impact of data compression on embedded meters, especially the STM32WBA, provides valuable guidelines for optimizing device performance without compromising information integrity or operational stability. By demonstrating the viability of HEATSHRINK as an efficient solution for reducing the volume of data transmitted, this study makes it possible to implement more agile and energy-efficient systems, extending the life of the meters and optimizing the consumption of computing resources. In addition, the proposed configurations allow Wasion's devices to be better adapted to low-bandwidth communication networks such as LoRaWAN and NB-IoT, extending the company's reach and competitiveness in the global smart metering market. Based on the results presented, future integrations with advanced techniques such as hybrid compression and machine learning could further improve the efficiency of the company's embedded systems, consolidating its capacity for innovation and meeting the growing demands for sustainable and scalable solutions in the energy sector.

Recommendations for future work include exploring hybrid compression techniques that combine classic algorithms, such as those used in this study, with modern methods based on machine learning. Approaches such as compressed neural networks and adaptive predictive models could be investigated to further optimize compression efficiency, especially on devices with severe resource constraints. These solutions have the potential to identify patterns in real time and adapt the compression process to the dynamic characteristics of the data collected, providing greater flexibility and efficiency. We can also explore a practical comparison with equipment in the field, analyzing efficiency before and after implementing the algorithm.

In addition, we recommend investigating advanced memory management strategies for embedded devices with even more limited resources than the STM32WBA, as well as new sample configurations and collection times. Methods such as dynamic memory allocation and segmented buffers could be tested to offer greater flexibility in resource management during data compression and sending. This line of research could include the development of detailed metrics to evaluate the impact of compression on overall system performance, providing a more comprehensive view of the benefits and limitations of the proposed solutions.



Another area of interest would be the integration of compression techniques with monitoring and predictive analysis systems, used in applications such as predictive maintenance and anomaly detection. Combining efficient compression with predictive models would not only reduce the volume of data transmitted but would also allow faults to be anticipated and anomalous behavior to be identified with greater precision. This approach would be particularly useful in industrial IoT networks, where reliability and efficiency are crucial.

Finally, field validation of the optimized configurations proposed in this study is a fundamental step in consolidating the theoretical and experimental results. Future studies could carry out large-scale tests, simulating different operating conditions and assessing the robustness, efficiency and reliability of the proposed solutions in real scenarios. This type of practical validation has the potential to expand the applications of compression techniques in sectors such as energy, health and transport, broadening the impact and relevance of the contributions of this work.

ACKNOWLEDGEMENTS

We would like to thank Wasion Group, which believed in Evolução Instituto and its partners, NEPEN and IFRO, to carry out this research and prototype the solution found. This research was funded by Lei nº 8.387/1991 (Zona Franca de Manaus) Framed in articles 21 and 22 of Decree No. 10,521, of 10/15/2020



REFERENCES

1. BURROWS, Michael; WHEELER, David J. A Block-Sorting Lossless Data Compression Algorithm. *Digital Equipment Corporation*, n. 124, 1994. Disponível em: <https://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf>.
2. CLEARY, John G.; WITTEN, Ian H. Data Compression Using Adaptive Coding and Partial String Matching. *IEEE Transactions on Communications*, v. 32, n. 4, p. 396–402, abr. 1984.
3. COSTANZI, Jean Eduardo; GUEMBAROVSKI, Ricardo Haus. Aplicação de Técnicas de Mineração de Dados para Aprimoramento da Gestão do Sistema Elétrico de Baixa Tensão. *Seminário Nacional de Distribuição de Energia Elétrica (SENDI)*, 2023.
4. JÚNIOR, Javan Ataíde de Oliveira; CAMARGO, Edson Tavares de; OYAMADA, Márcio Seiji. Data Compression in LoRa Networks: A Compromise between Performance and Energy Consumption. *Journal of Internet Services and Applications*, v. 14, n. 1, p. 1–15, 2023.
5. JÚNIOR, Javan Ataíde de Oliveira; OYAMADA, Márcio Seiji. Avaliando o impacto da compressão de dados no desempenho e energia em redes LoRa. *Seminário Nacional de Distribuição de Energia Elétrica (SENDI)*, 2023.
6. KETSHABETSWE, Keleadile Lucia; ZUNGERU, Adamu Murtala; MTENGI, Bokani; LEBEKWE, Caspar K.; PRABAHARAN, S. R. S. Data Compression Algorithms for Wireless Sensor Networks: A Review and Comparison. *IEEE Access*, v. 9, p. 136872–136882, set. 2021. Acesso em: 25 jul. 2024.
7. KNUTH, Donald E. *The Art of Computer Programming: Sorting and Searching*. v. 3. Addison-Wesley, 1998.
8. KODITUWAKKU, S. R.; AMARASINGHE, U. S. Comparison of Lossless Data Compression Algorithms for Text Data. *Indian Journal of Computer Science and Engineering*, v. 1, n. 4, p. 416–426, out. 2010. Acesso em: 18 abr. 2024.
9. LEE, Jaemoon; GONG, Qian; CHOI, Jong; BANERJEE, Tania; KLASKY, Scott; RANKA, Sanjay; RANGARAJAN, Anand. Error-Bounded Learned Scientific Data Compression with Preservation of Derived Quantities. *Applied Sciences*, v. 12, n. 13, p. 6718, jul. 2022. Acesso em: 02 ago. 2024.
10. LEE, Wen-Han; CHEN, Min-Hua; LEE, Chen-Yu; LI, Yu-Liang. Lossless Compression of Data Tables in Mobile Devices using Co-clustering. 2016. Acesso em: 12 ago. 2024.
11. MALANDRINO, Francesco; DI GIACOMO, Giuseppe; KARAMZADE, Armin; LEVORATO, Marco; CHIASSERINI, Carla Fabiana. Tuning DNN Model Compression to Resource and Data Availability in Cooperative Training. *IEEE/ACM Transactions on Networking*, v. 32, n. 2, p. 1600–1615, abr. 2024. Acesso em: 18 abr. 2024.
12. MEFFE, André; PRIETO, Mauricio Andrés Paez; GARCEZ NETO, Alvaro de Freitas; TEODORO JUNIOR, José Raimundo. Solução de baixo custo para leitura e gerenciamento remoto de unidades consumidoras dispersas com tecnologia LoRaWAN. *Seminário Nacional de Distribuição de Energia Elétrica (SENDI)*, 2023.



13. MOON, Aekyeung; KIM, Jaeyoung; ZANG, Jialing; SON, Seung Woo. Evaluating Fidelity of Lossy Compression on Spatiotemporal Data from an IoT. *Computers and Electronics in Agriculture*, v. 154, p. 304–313, nov. 2018. Acesso em: 19 ago. 2024.
14. PYTHON SOFTWARE FOUNDATION. bz2 Compression compatible with bzip2. 2024. Disponível em: <https://docs.python.org/3/library/bz2.html>.
15. QIN, Jiancheng; LU, Yiqin; ZHONG, Yu. Block-Split Array Coding Algorithm for Long-Stream Data Compression. *Journal of Sensors*, v. 2020, p. 1–22, 2020. Acesso em: 22 jul. 2024.
16. RIBERA NETO, Danilo; LEITE, Marcelo Antonio Ramos. Aplicação de Inteligência Artificial no Processo de Análise dos Dados de Medição com Foco na Recuperação de Energia. *Seminário Nacional de Distribuição de Energia Elétrica (SENDI)*, 2023. Acesso ao texto completo disponível no PDF fornecido pelo usuário.
17. SANDOVAL, S.; VARGAS, A.; ORTEGA, C.; VIDAL, Y. Three-way unsupervised data mining for power system applications based on tensor decomposition. *International Journal of Electrical Power & Energy Systems*, v. 123, p. 106241, out. 2020. Acesso em: 12 ago. 2024.
18. SANTOS, Willian Garcia Viega dos; QUEIROZ COSTA, Juliana Vitória de; SALUSTIANO DE OLIVEIRA, Rodrigo; CABRAL, Mayara de Oliveira. Medidores Inteligentes com Tecnologia NB-IoT: Análise em Ambientes Urbanos e Regiões Rurais com o uso de Blindagem. *Seminário Nacional de Distribuição de Energia Elétrica (SENDI)*, 2023.
19. SEWARD, Julian. bzip2 and libbzip2: A program and library for data compression. 2024. Disponível em: <https://www.sourceware.org/bzip2/>.
20. ZHANG, Fang; LIU, Meiqian; ZHANG, Zixuan; HE, Jinghan; GAO, Wenzhong. Real-time Synchronphasor Data Compression Technique with Phasor Interpolation and Extrapolation. *Journal of Modern Power Systems and Clean Energy*, v. 11, n. 3, p. 803–815, maio 2023. Acesso em: 18 abr. 2024.
21. ZHANG, Zhihua. The improvement of the discrete wavelet transforms. *Mathematics*, v. 11, n. 1770, 2023. Acesso em: 02 ago. 2024.