Statistical and Entropic Modeling for Lossless Data Compression in the Internet of Things

A PREPRINT

Sébastien Philippe
Independent Researcher, Private Research
Nîmes, France
dondenathea@gmail.com

April 25, 2025

ABSTRACT

This paper introduces a novel lossless compression scheme designed specifically for highly constrained environments such as those encountered in the Internet of Things (IoT). The proposed method, termed *Chaotic Compression Standard (CCS)*, employs a hierarchical symbol partitioning strategy coupled with a simplified variant of the range Asymmetric Numeral Systems (rANS), effectively avoiding costly division operations. A key contribution of CCS lies in the dynamic embedding of zone-identifying bits within the compressed data stream, achieved through precise alignment within fixed-size binary packets.

This architecture offers a compelling balance between compression efficiency, low computational complexity, and optimal suitability for embedded systems.

Empirical results indicate that CCS achieves slightly better compression ratios (approximately 1%) compared to classical methods such as rANS coding, but offers a substantial improvement in encoding speed, reducing processing time by approximately 80%. These results highlight its particular suitability for embedded IoT applications.

Given the exponential growth in data generated by IoT devices, conventional lossless compression methods are increasingly constrained by processing and energy limitations. CCS addresses these challenges by integrating statistical modeling and an entropy coding paradigm capable of outperforming traditional approaches such as Huffman, range coding, and arithmetic coding. Preliminary evaluations underscore the algorithm's effectiveness, particularly on structured datasets, and highlight its strong potential for hardware-level deployment and future entropy-aware optimizations.

Keywords Lossless data compression, Entropy coding, Symbol frequency analysis, Information theory.

1 Introduction

Videos, photos, connected devices... The explosion of digital data creation, driven by the rise of artificial intelligence, social networks, the Internet of Things (IoT), and Big Data, presents major challenges in terms of storage, transmission, and processing. According to Statista [1], the volume of data generated worldwide is expected to exceed 394 zettabytes by 2028, up from 149 zettabytes in 2024.

As a result, energy consumption has soared. In France, a study [2] estimated digital-related electricity consumption at 56.5 TWh in 2020, broken down into user devices (43 TWh), network infrastructure (3.5 TWh), and data centers (10 TWh). Today, digital technologies account for approximately 2.5% of France's carbon footprint [3].

Despite improvements in the energy efficiency of data centers—through cooling optimization, the use of renewable energy, and the design of more efficient servers—as well as the development of less power-hungry devices, optimizing data processing remains a critical challenge, as digital sobriety is far from being a reality.

In this context, data compression, and more specifically lossless compression, plays a crucial role in optimizing resource usage. Our experimental results demonstrate that the proposed CCS algorithm achieves a modest improvement in compression efficiency (about 1%), but significantly reduces encoding time by approximately 80% compared to traditional methods, making it especially relevant for constrained IoT environments.

Data compression, or source coding, is a computational operation that transforms a sequence of bits A into a shorter sequence of bits B during compression, such that the original information—or a close approximation—can be restored using a decompression algorithm. It is a coding operation that reduces the size of data (for transmission or storage) at the cost of a computational effort.

There are two main types of data compression:

1. Lossy compression (e.g., JPEG, MP3, DivX)
2. Lossless compression (e.g., archives, executable files, texts)

The proposed research topic falls within the scope of lossless digital data compression.

This research project aims to develop an innovative data compressor, named CCS, which leverages a statistical analysis approach combined with entropy-based coding applied to data sets. This approach differs from traditional methods by focusing on more efficient identification and representation of information, enabling a reduction in file size without losing critical data.

More specifically, this research is centered on the Internet of Things (IoT), a market undergoing exponential growth and profoundly transforming everyday life. IoT devices are becoming ubiquitous in our homes, businesses, and cities, creating countless opportunities across various sectors such as industry, healthcare, agriculture, and smart home automation. This growth is fueled by technological advancements, decreasing costs of sensors and networks, and the increasing adoption of cloud computing. However, this expansion also raises important challenges, including the massive increase in data volume, data security, and privacy protection.

In this context, the need for data compression in such systems has become an absolute necessity in light of the explosive growth in their numbers and the volumes of data they generate. Technical constraints—such as limited processing power, RAM, and sequential operation—associated with microcontrollers (integrated circuits that embed a processor, memory, and I/O), as well as IoT network protocols (e.g., Bluetooth Low Energy, Message Queuing Telemetry Transport, ZigBee, Near Field Communication, Long-range wide-area network…), which often have low bandwidth and energy constraints, make compression indispensable.

Reducing data size helps optimize transmission, lower communication costs, and extend device battery life. Furthermore, data compression is essential for the efficient storage of this information in the cloud or in local databases, to enable easier processing and analysis.

This document primarily focuses on the encoding process and will only briefly address decoding. The latter is of less importance in the context of IoT, as it is typically handled by servers with significantly greater resources.

2 State of the Art

2.1 Entropy of a source S

In information theory [4], the entropy of a source is the average amount of information produced per emitted symbol, measured in bits. It represents the theoretical lower bound for the lossless compression rate.

- ***Methodology***

$$H(S) = \sum_{P(x_i)>0} P(x_i) \log_2 \left(\frac{1}{P(x_i)}\right) \quad (1)$$

$$H(S) = -\sum_{P(x_i)>0} P(x_i)\, log_2\big(P(x_i)\big) \quad (2)$$

With: x_i a symbol and $P(x_i)$ its probability of occurrence.

- ***Example***

Entropy coding (or Variable-Length Statistical Coding) refers to the process of encoding symbols—produced by the modeling stage—using variable-length codes, depending on their probability of occurrence. Symbols that appear more frequently are assigned shorter codes, while rarer symbols receive longer ones.

As an illustrative example, we will use the sentence: “this is an example of a huffman tree”, which will be referenced throughout this document.

In this case, the sentence contains 36 characters in total, with 16 unique symbols (including spaces and repeated letters). The entropy of the source, based on the probability distribution of the characters, is therefore:

$$36 \times \frac{\ln 16}{\ln 2} = 144 \text{ bits}$$

2.2 Huffman coding

Huffman coding [5] is a method for constructing an optimal binary tree (in the sense of minimizing the weighted average code length) based on the frequencies of the symbols.

- ***Methodology***

1. Build a forest of single-node trees, one for each symbol.
2. Iteratively merge the two trees with the lowest probability until only one tree remains.
3. Assign binary digits (0 and 1) to the left and right branches recursively.

The average code length is given by: $Lavg = \sum_{i=1}^{n} P(xi) \times l(xi)$ where $l(xi)$ is the length of the codeword assigned to symbol xi. Huffman coding guarantees the following bound: $H(S) \leq Lavg < H(S) + 1$.

- ***Example***

Construction of the Tree Using “this is an example of a huffman tree”:

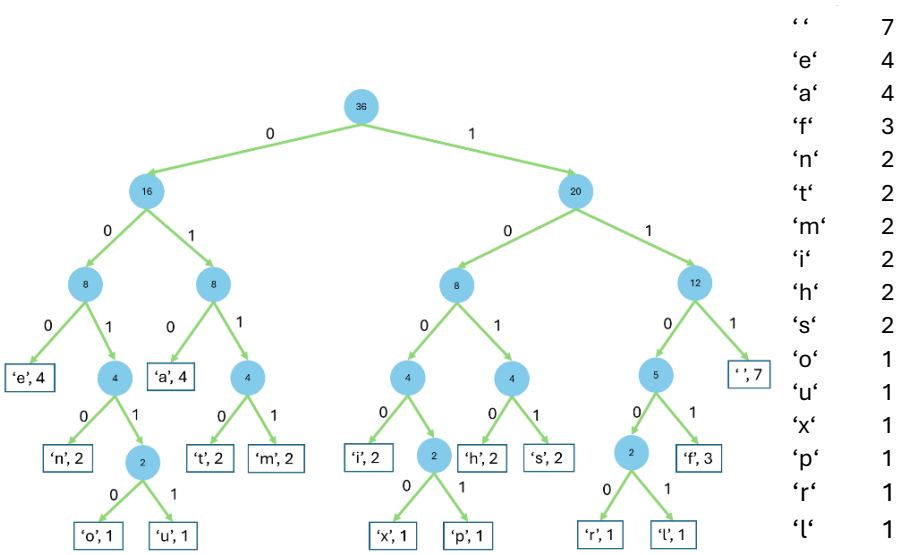

Figure 1: Huffman tree (left) and frequencies (right)

Each branch in the tree is associated with a marking bit, resulting in a total of 135 bits, compared to the 288 bits required in the original uncompressed representation (assuming 8 bits per character for 36 characters). This coding method is highly efficient and extremely fast, and it has been widely used since its discovery.

2.3 ANS (Asymmetric Numeral Systems)

ANS coding [6] belongs to a family of fast and efficient entropy coders. It encodes the state of a symbol stream into a single natural number. This model can be implemented in different variants:

1. **rANS** (*Range ANS*): integer-based encoding with base-2 normalization, and
2. **tANS** (*Table ANS*): pre-generated transition tables.

The average code length tends toward: $Lavg \approx H(S) \pm \varepsilon$ where ε is a small deviation depending on implementation details and symbol statistics.

- ***Methodology***

In simplified ANS encoding, there is a relationship between:

1. the current state x,
2. The size of the normalization table L, and
3. the probability of the symbol s.

The general formula for encoding a symbol s is:

$$x_{new} = L \times \left\lfloor \frac{x}{f(s)} \right\rfloor + Index(s, Table\ ANS) \quad (3)$$

Where :

1. $f(s)$ is the frequency of the symbol s (scaled relative to a total L, typically a power of 2 for normalization purposes), and
2. $Index(s, Table\ ANS)$ is the index associated with the symbol s in the ANS transition table.

- ***Example***

Let's return to the example: "this *is an example of a huffman tree"* with $x_{start} = 8388608$ et $Scale_{frequencies} = 14\ bits$.

$$fs' = round(ps \times scale) \quad (4)$$

$$fs' = round\left(\frac{f_s \times\ scale_{frequencies}}{N}\right) \quad (5)$$

$$x_{new} = \lfloor freqx \rfloor \times 2^{14} + (xmodfreq) + x_{start} \quad (6)$$

Let's compute for our sequence:

Fréquencies of the 36 characters:

Symbol	Frequency	Index	Normalized Frequency
' '	7	29	3189
'e'	4	25	1820
'a'	4	21	1820
'f'	3	18	1350
'n'	2	16	910
't'	2	14	910
'm'	2	12	910
'i'	2	10	910
'h'	2	8	910
's'	2	6	910
'o'	1	5	455
'u'	1	4	455
'x'	1	3	455
'p'	1	2	455
'r'	1	1	455
'l'	1	0	455

Figure 2 : Table of Normalized Frequencies

Symbol 1 : 'e' (*Encoding starts from the end)*

$$x_{new} = \left\lfloor \frac{8388608}{1820} \right\rfloor \times 16384 + (8388608\ mod 1820)\ +\ 5009) = 75519093$$

Symbol 2 : 'e'

$$x_{new} = \left\lfloor \frac{75519093}{1820} \right\rfloor \times 16384 + (75519093 \; mod \; 1820) \; + \; 5009) = \; 679842718$$

Symbol 3 : 'r'

$$x_{new} = \left\lfloor \frac{679842718}{455} \right\rfloor \times 16384 + (679842718 \; mod \; 455) \; + \; 5009 = 2592$$

Etc…

Symbol 36 : 't'

$$x_{new} = \; 1495248540041779227644356004603921659811643052 56$$

Thus, we obtain approximately 157 bits in base 2.

2.4 Compatibility issues

These algorithms each have their own advantages and drawbacks in terms of compression ratio, processing time (compression/decompression), required resources, and portability.

In the context of IoT systems, devices are often battery-powered and subject to strict energy constraints. Although advanced entropy coders like ANS achieve excellent compression, they typically involve computationally costly operations such as division or complex state renormalizations. Such operations significantly impact battery life and real-time responsiveness in IoT devices. The CCS algorithm specifically addresses these shortcomings by eliminating divisions and employing simpler arithmetic operations, thus ensuring compatibility with resource-constrained embedded systems.

3 Detailed Presentation of the CCS Approach

3.1 Encoding

The CCS approach is structured into several key steps:

3.1.1 Step 1 : Preprocessing and Hierarchical Partitioning

A. Frequency estimation and preliminary sorting

As in many entropy-based algorithms, the first step consists in retrieving the symbol frequencies from the input sequence.

- ***Methodology***

  Let $X = (x1, x2, \ldots, xN)$ be a vector of observed symbols. The frequency $f(xi)$ of a symbol x is computed as:

$$f(x) = \frac{\#\{i \mid xi = x\}}{N} \text{ for all } x \in \Sigma \tag{7}$$

  where Σ is the alphabet of symbols present in the sequence. This allows us to associate each symbol with its empirical probability of occurrence.

  Next, to prepare for optimal partitioning or efficient encoding, the set of symbols $x \in \Sigma$ is sorted in descending order of frequency:

$$x(1), x(2), \ldots, x(n) \text{ with } f(x(1)) \geq f(x(2)) \geq \cdots \geq f(x(n))$$

This sorting step is crucial for the construction of the next stage. It also helps identify homogeneous frequency zones, which enables more intelligent and adaptive partitioning strategies.

- ***Example***

  We continue with the example: "this is an example of a huffman tree". The symbol frequencies are identical to those used earlier in the Huffman coding section.

B. Recursive Partitioning Based on Entropic Efficiency

This recursive partitioning is based on the median of symbol frequencies, in order to create zones with statistically homogeneous distributions.

Each partition generates a hierarchical binary encoding, where upper-frequency zones are encoded with a "1", and lower-frequency zones with a "0".

- ***Methodology***

  Let $S = \{x1, x2, \ldots, xn\}$ be a set of symbols with an associated frequency function $f(xi)$. The goal is to find a partition $P = \{S1, \ldots, Sk\}$ such that the total compression cost is minimized.

  For each subset S, we define:

  1. $Nj = \mid Sj \mid$, as the number of symbols in the subset,
  2. $H(Sj)$, as the local entropy,
  3. Bj represents the number of bits required to identify zone Sj among the different created partitions, typically estimated as $log2(k)$, where k is the number of zones,
  4. $C(Sj) = Nj \cdot H(Sj) + Bj$, as the total cost, where Bj is the number of additional bits needed to identify the zone Sj.

  The recursive division by the median mmm of the frequencies in Sj yields two subsets:

  $SL = \{xi \in Sj \mid f(xi) \leq m\}$ and $SH = \{xi \in Sj \mid f(xi) > m\}$

  This subdivision is accepted if:

  $$C(SL) + C(SH) + Bsplit < C(Sj)$$

  In other words, the subdivision is retained only if the entropy gain outweighs the structural cost $Bsplit$. This rule ensures that partitioning actually improves compression, rather than structurally degrading it.

  Finally, a useful global metric is the product of the sizes of all partitions, which may be used to estimate statistical dispersion or combinatorial complexity:

  :

  $$\prod P = \prod_{j=1}^{k} Nj \tag{8}$$

- ***Example***

  In Figure 3, the symbol with rank 13 has a binary suffix of 111. Its encoded value is therefore 111 + 10, representing its position within the remaining interval.

  When the partitioning no longer yields a meaningful gain in terms of entropy reduction, the process is stopped.

  Each frequency is thus associated with a unique binary sequence, allowing for unambiguous identification.

  At this stage, it is possible to encounter a longer binary sequence for a higher frequency, particularly in cases where the distribution is more uniform. In such scenarios, it becomes relevant to process these sequences using a Huffman tree, in order to find the optimal encoding.

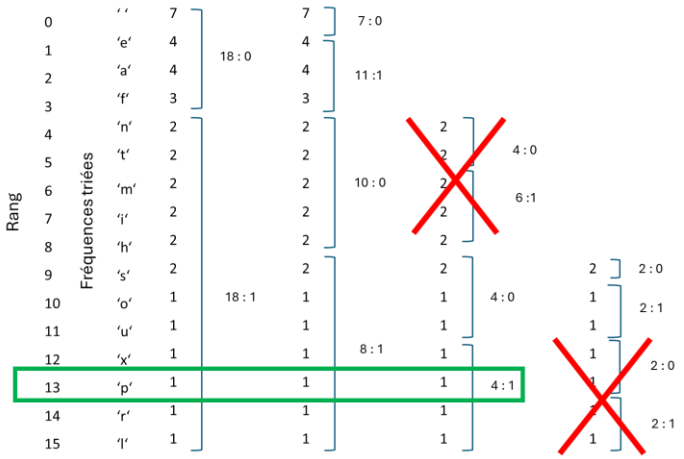

Figure 3 : Partitioning of Frequencies by Median

We observe here that the second interval contains 3 items, which should normally be encoded using 2 bits, since $2^1 < 3 < 2^2$. Similarly, the third interval, with 5 values, should ideally be encoded using 3 bits.

This inefficiency can have significant consequences in terms of compression ratio. In this example, for the second interval, we would have 11 symbols encoded with 2 bits each, resulting in 22 bits, whereas the actual information content should be: $\frac{\ln(3)}{\ln(2)} \approx 1.584$ bits per symbol. Thus leading to a loss of approximately 4 bits. For the third interval, the inefficiency amounts to about 6 bits.

3.1.2 Step 2: Encoding Positions Within Intervals

This encoding step is designed to address the inefficiency discussed previously. The idea is straightforward and draws inspiration from the ANS encoder: transform a sequence of symbols into a large decimal value, which is then encoded in chunks of 32 or 64 bits.
However, the comparison with ANS ends there. The proposed CCS compressor simplifies the ANS-like approach by eliminating costly operations such as integer division and floor functions, which are time-consuming on embedded processors.
Instead, simple multiplications and additions are used to store the values efficiently.

- ***Methodology***

Let x be a symbol encoded according to its rank within the previously defined partition. The overall method consists in building a single large decimal value representing the sequence:

1. $Nj = | Sj |$ as the number of symbols in partition j,
2. x_m as the current symbol, and
3. Bj represents the number of bits required to identify the zone Sj

Then,

$$x_{codé} = \sum_{m=1}^{k} \left(x_m \times \prod_{j=m+1}^{k} Nj \right) \tag{9}$$

$$B_{total} = \sum_{j=1}^{k} Bj \tag{10}$$

Since the encoded number can become extremely large, it is necessary—particularly in the context of IoT systems—to limit its size. Two methods are possible.

The first is the approach used in ANS encoding, which involves regularly flushing the compressed state by emitting the least significant bits in fixed-size blocks (e.g., 32 or 64 bits) whenever the state exceeds a certain threshold.
This renormalization mechanism ensures that the state remains within a controlled range while preserving the reversibility of the encoding process. Each time the state becomes too large, a portion is flushed to the binary output, and the remaining part is retained to process the subsequent symbols.

The second method consists in explicitly segmenting the value so that it fits within a 32- or 64-bit variable, depending on the capabilities of the microcontroller being used.
This simpler approach is particularly well-suited to resource-constrained embedded architectures.

For our purposes, we will initially adopt the second method.
Since each interval may have a different size, it is important to ensure that the cumulative decimal value generated during encoding remains below 32 bits (or 64, depending on the case).

$$I_{codée} = \prod_{i=m+1}^{k} Nj < 2^{32} \tag{11}$$

- ***Example***

Continuing with the previous example, “this is an example of a huffman tree”:

$$
\begin{aligned}
\text{‘ ‘} &= 7 * (2\ bits + \ln(1)/\ln(2)) = 14\ bits \\
\text{‘e‘} &= 4 * (2\ bits + \ln(3)/\ln(2)) = 14.34\ bits \\
\text{‘a‘} &= 4 * (2\ bits + \ln(3)/\ln(2)) = 14.34\ bits \\
\text{‘f‘} &= 3 * (2\ bits + \ln(3)/\ln(2)) = 10.75\ bits \\
\text{‘n‘} &= 2 * (2\ bits + \ln(5)/\ln(2)) = 8.64\ bits \\
\text{‘t‘} &= 2 * (2\ bit + \ln(5)/\ln(2)) = 8.64\ bits \\
\text{‘m‘} &= 2 * (2\ bit + \ln(5)/\ln(2)) = 8.64\ bits \\
\text{‘i‘} &= 2 * (2\ bit + \ln(5)/\ln(2)) = 8.64\ bits \\
\text{‘h‘} &= 2 * (2\ bit + \ln(5)/\ln(2)) = 8.64\ bits \\
\text{‘s‘} &= 2 * (4\ bit + \ln(1)/\ln(2)) = 8\ bits \\
\text{‘o‘} &= 1 * (4\ bit + \ln(2)/\ln(2)) = 5\ bits \\
\text{‘u‘} &= 1 * (4\ bit + \ln(2)/\ln(2)) = 5\ bits \\
\text{‘x‘} &= 1 * (3\ bit + \ln(4)/\ln(2)) = 5\ bits
\end{aligned}
$$

Figure 4: Number of bits required per symbol

Thus, we obtain approximately 134.63 bits in base 2.

- ***Optimization***

1. The partitioning strategy presented here is based on a binary split using the median. However, in certain cases, a three-way partition may yield better results.
2. Another optimization strategy would involve adapting the compression process to the actual frequency distribution.
   When the symbol distribution is relatively uniform, the partitions generated by the CCS algorithm are nearly equal in size. In the case of balanced frequency distributions, the CCS approach naturally reduces to a uniform fixed-length coding, similar to a trivial Huffman encoding, where each symbol is identified by its position using $\lceil log2(N) \rceil$bits.
3. To better adapt to local variations in the data, it is also possible to periodically recompute the observed symbol frequencies during the encoding process.
   This approach allows for dynamic adjustment of the partitioning based on local statistical changes, thereby improving overall compression efficiency.
   In practice, after processing a certain number of symbols, the frequencies can be recalculated, and a new optimized partitioning can be applied.
   This technique, inspired by adaptive methods such as Adaptive Huffman Coding or Dynamic ANS, helps maintain compression rates close to the theoretical entropy even for heterogeneous or evolving data sources.

In summary, the CCS approach distinguishes itself from traditional entropy-based methods (Huffman, rANS, Arithmetic Coding) by combining hierarchical partitioning with simplified arithmetic operations, thus avoiding costly divisions. This strategy results in a slight improvement in compression performance, accompanied by a significant encoding speed gain of about 80%, making CCS particularly beneficial for resource-limited IoT systems.

3.2 Décoding

The decoding process aims to recover the original symbols from the encoded frequencies, the zone bit sequences, and the position values within the intervals.

The first step is to reconstruct the partition intervals based on the frequencies encoded in the data stream. This reconstruction uses the same algorithm as in the encoding phase.

From the zone bit sequences, the decoder determines the size of the associated interval and then recovers the encoded symbol by dividing the decimal value by the size of that interval.

While decoding is slightly more complex than encoding, it is typically handled by servers, which have the computational resources to perform these operations efficiently.

- ***Methodology***

Let $x_{new\_encoded}$ be the encoded decimal value, x_m the sequence of symbols to be recovered, ordered by their rank in the previously determined partition, and I_m, the product of the remaining interval sizes. The general method to decode from a single large decimal value is as follows:

$\forall m \in \{0, \dots, k\}$,

$$x_m = \left\lfloor \left(\frac{(x_{new\_encoded})}{I_m}\right) \right\rfloor \qquad \text{where} \quad I_m = \prod_{i=m+1}^{k} Nj$$

$$x_{new_{encoded}} = \left(x_{new_{encoded}} \; mod \; I_m\right) \times I_m \tag{12}$$

Each symbol is recovered by successively dividing and taking the remainder with respect to the corresponding interval sizes, in reverse order of encoding. This process mirrors a mixed-base number system (similar to variable-radix decoding), where each base corresponds to the size of a partition or sub-interval.

3.3 Comparatif

In the end, here is the comparative table of the methods studied:

Tableau 1: Comparative Table of Compression Ratios

	Initial size	Maximal entropy	Huffman coding	rANS coding	CCS coding
"this is an example of a huffman tree"	288 bits	144 bits	135 bits	157 bits	134.63 bits

In this specific case, Huffman coding is very effective because the frequencies are well aligned with powers of 2.

- ***Innovative* Contribution of the Simplified CCS Encoding**

This variant differs significantly from traditional approaches by greatly reducing algorithmic complexity. This design explicitly avoids costly operations such as real-time divisions or subtractions, making it particularly well-suited for low-power microcontrollers typically found in IoT devices.

Additionally, a structural enhancement is proposed: to exploit the zone bits (generated from hierarchical partitioning) as adaptive padding elements.
When, after encoding, the compressed state reaches a bit length not aligned with a full byte (e.g., 29 bits), zone bits are selectively inserted to complete the data up to 32 bits before writing.

This ensures that the binary stream remains byte-aligned while seamlessly embedding hierarchical information without overhead.

A zone bit queue mechanism is introduced, which is progressively filled and flushed into the available spaces at the appropriate time. This management allows maintaining a single unified stream (positions + zones), perfectly compressed and decodable, without any physical separation or loss of information.

- ***Experimental Protocols***

To validate the approach experimentally, a small software tool implementing both encoding and decoding has been developed.

The initial comparison focuses on raw performance metrics, including compression ratio, compression time, and decompression time.
In a subsequent phase, additional data such as resource usage (CPU time, RAM consumption) and the code size of the algorithm will be analyzed.

This second evaluation phase will be carried out once the developed software has been sufficiently optimized.

These preliminary tests are conducted in single-threaded mode on a laptop with an Intel i3 processor and 8 GB of RAM, running Windows 10, using Visual Studio 2022 as the development environment.

The first tests were performed on a panel of six files sourced from a well-known dataset: the Canterbury Corpus, a collection of varied English text files [4]:

- alice29.txt, text, english text, 152089 bytes
- asyoulik.txt, play, shakespeare, 125179 bytes
- fields.c, Csrc, C source, 11150 bytes
- Ice10.txt, tech, technical writing, 426754 bytes
- Plrabn12.txt, poem, Poetry, 481861 bytes
- Ptt5, fax, CCITT test set, 513216 bytes

- ***Results***

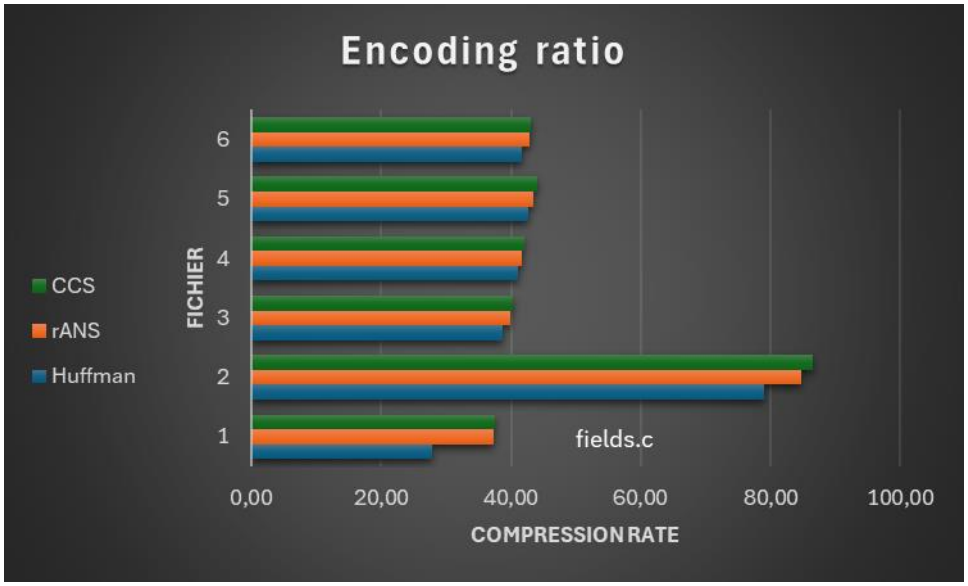

Figure 5 : Encoding ratio

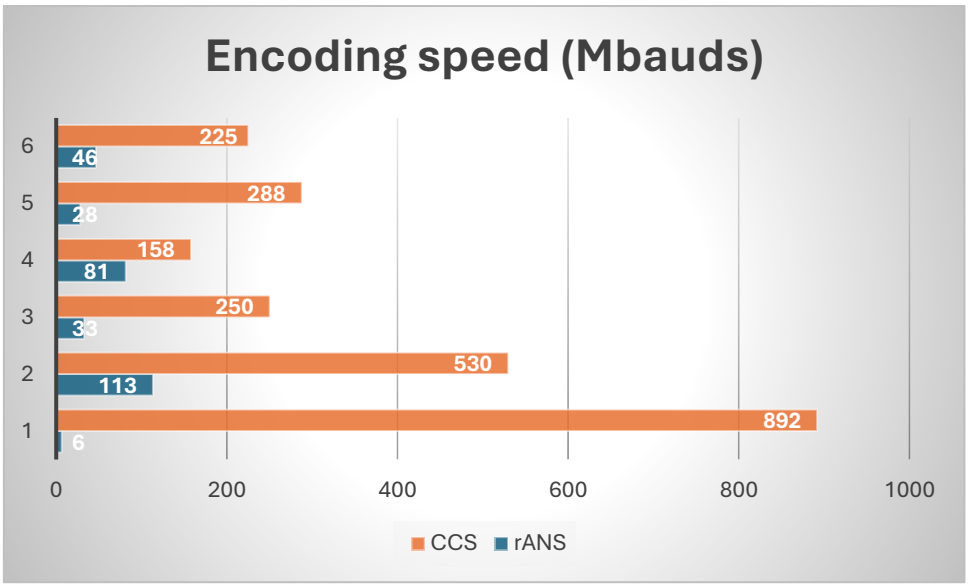

Figure 6 : Encoding speed

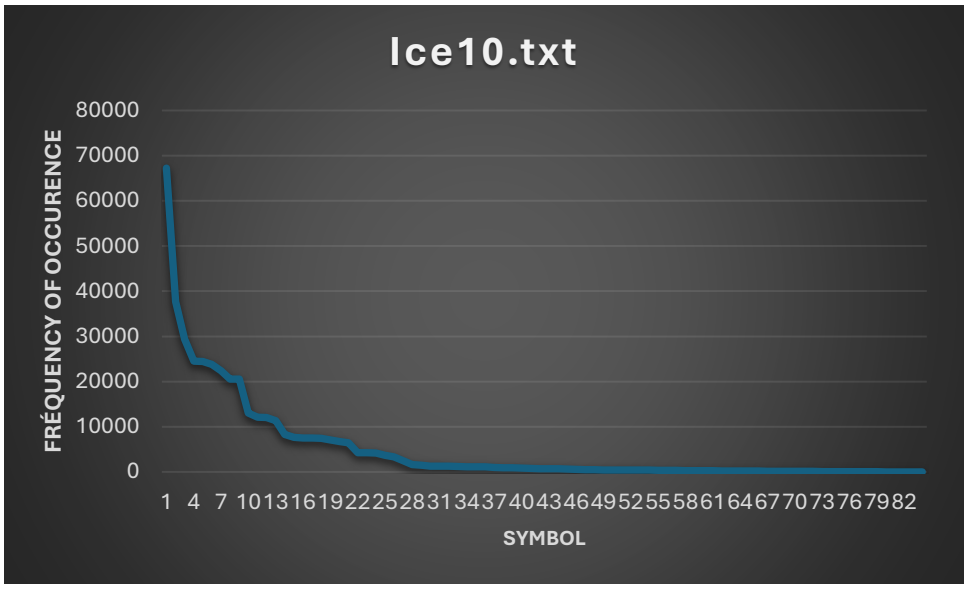

Figure 7 : Example of Frequency Distribution for the File Ice10.txt

- ***Interpretation of results***

Compared to established techniques such as Huffman coding, rANS, Range Coding, or Arithmetic Coding, CCS demonstrates a competitive compression-time trade-off, particularly on datasets with strong structural coherence.
When compared specifically to rANS, CCS achieves equivalent or superior compression ratios while maintaining higher compression speeds.

This balanced performance positions CCS as a serious alternative for IoT applications, where speed and low energy consumption are critical requirements.
However, for highly heterogeneous or high-entropy datasets, algorithmic improvements will be necessary to maintain a competitive advantage.

4 Current Advantages and Limitations

The results show that the Chaotic Compression Standard (CCS) approach—based on hierarchical partitioning and simplified encoding—is highly promising for lossless compression in resource-constrained environments such as the Internet of Things (IoT).

The CCS algorithm offers an optimal balance between high compression ratio and low computational complexity, significantly reducing both energy consumption and memory usage compared to classical methods.

The main advantages of CCS include:

1. High efficiency on structured data, where it often outperforms established algorithms such as Huffman, Range Coding, or Arithmetic Coding.
2. The elimination of costly operations such as division, replaced by simple multiplication and addition, making it especially suited for low-power microcontrollers.

However, CCS demonstrates slightly better but comparable compression efficiency relative to classical methods, especially on highly structured data sets. The primary advantage evidenced by our results is the substantial reduction in encoding time (approximately 80%). On the other hand, for highly heterogeneous or random datasets, additional optimization might be necessary to preserve this advantageous performance.

- ***Future Work and Research Directions***

Algorithm Optimization, Memory Management, and Adaptation to Low-Power Environments. In a second phase, through the use of an exponential pseudo-randomization algorithm, several files containing a fixed number of unique symbols with varied frequency characteristics will be generated.

For example:

General Equation:
$y = n * e^{-k*x}$
$n : fréquence\ de\ symboles\ maximale$
$k : coefficient\ de\ décroissance$

Discrete Equation:
$f(i) = decay\_factor^{i}$
$decay\_factor^{i} = e^{i*\ln(decay\_factor)}$

In addition, practical integration of CCS into existing IoT platforms, through standardized communication protocols or embedded software frameworks, constitutes a promising avenue. Evaluations in real-world IoT deployments (e.g., smart sensor networks or wearable health devices) will further validate the applicability and robustness of CCS in constrained operational environments.

- ***Algorithm Optimization, Memory Management, and Adaptation to Low-Power Environments***

The next steps should focus on fine-tuning the CCS algorithm, particularly in terms of memory management and further energy consumption reduction. Approaches such as parallelization and hardware implementation (e.g., FPGA or ASIC) should be explored to maximize efficiency on microcontrollers with limited resources.

As an example, the encoding process may generate the following transmission frame for a simplified data stream:

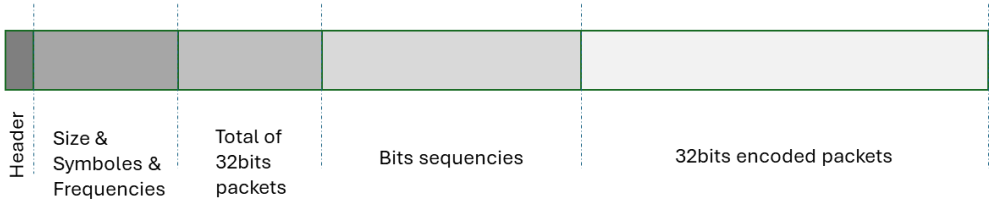

Figure 8 : Example of a Possible Frame

- ***Potential Exploration of Cryptographic Applications Based on Chaotic Systems***

The inherently chaotic nature of the CCS model suggests promising opportunities for its adaptation to cryptographic applications, particularly in the context of post-quantum cybersecurity. Its chaotic properties could be leveraged to generate complex and robust keys, resistant to quantum attacks, thus opening up exciting interdisciplinary research directions that bridge data compression and embedded system security.

5 References

[1] https://www.statista.com/statistics/871513/worldwide-data-created/

[2] https://www.greenit.fr/etude-empreinte-environnementale-du-numerique-mondial/

[3] https://infos.ademe.fr/magazine-avril-2022/faits-et-chiffres/numerique-quel-impact-environnemental/

[4] Olivier Rioul, Julien Béguinot, Victor Rabiet, Antoine Souloumiac. La véritable (et méconnue) théorie de l'information de Shannon. 28e Colloque GRETSI'22, Sep 2022, Nancy, France. pp.908. hal-03718728

[5] Laurent Théry. Formalising Huffman's algorithm. [Research Report] Università degli Studi dell'Aquila. 2004. hal-02149909

[6] https://corpus.canterbury.ac.nz/descriptions/

[7] J. Duda, "Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding," arXiv:1311.2540, 2013.

[8] G. N. N. Martin, "Range encoding: an algorithm for removing redundancy from a digitized message," IBM UK Scientific Center, 1979.

[9] M. Young, The Technical Writer’s Handbook, University Science, 1989.

[10] P. Duchon, "Théorie de l'information," ENSEIRB, 2008-2009.

[11] M. Haj Taieb, "Codage arithmétique," Université Laval, 2012.