ASTES

Advances in Science, Technology and Engineering Systems Journal
Vol. 10, No. 5, 40-50 (2025)
www.astesj.com

ASTES Journal
ISSN: 2415-6698

CIRB-Edge for Secure, Energy-Efficient, and Real-Time Edge Computing

Mohamad Khalil Farhat*, Ji Zhang*, Xiaohui Tao, Tianning Li

University of Southern Queensland, Toowoomba, Queensland, Australia

ARTICLE INFO

Article history:
Received: 30 June, 2025
Revised: 29 August, 2025
Accepted: 01 September, 2025
Online: 19 September, 2025

Keywords:
Lossless integer compression
Edge computing
Changing Integer Representation and Base (CIRB)

ABSTRACT

In this work, we present CIRB-Edge, a novel integer compression method designed specifically to overcome the limitations of traditional techniques such as Huffman coding, Delta encoding, and dictionary-based algorithms. These legacy methods often fall short in meeting the stringent requirements of secure, energy-efficient, and real-time edge computing due to their high computational overhead, memory demands, or lack of integrated security features. CIRB-Edge addresses these challenges by introducing a lightweight, transformation-based compression scheme tailored for constrained hardware. By improving compression efficiency and reducing decoding latency, CIRB-Edge enables faster data processing and more efficient storage, which is particularly beneficial for edge computing and resource-constrained environments. Extensive experiments conducted on diverse edge platforms: Raspberry Pi 4, ESP32, and NVIDIA Jetson, demonstrate that CIRB-Edge consistently achieves compression ratios of up to 80%, while significantly outperforming existing state-of-the-art methods in throughput, energy efficiency, and security strength. These findings are supported by a series of experiments conducted on diverse datasets and IoT devices. This results in positioning CIRB-Edge as a practical and robust solution for next-generation edge computing applications requiring fast, secure, and low-power data processing.

1. Introduction

This paper is an extension of work originally presented at the International Conference on Signal Processing (ICSP) [1], where we introduced the initial design and evaluation of our compression approach (CIRB). In this extended version, we present CIRB-Edge, an enhanced and optimized version suitable for real-world edge devices, along with a more comprehensive evaluation.

The exponential growth of data in edge computing and Internet of Things (IoT) systems has increased the demand for efficient, real-time, and secure data compression [2]. Modern applications such as autonomous vehicles, smart healthcare, and industrial IoT generate vast streams of integer-based sensor data that must be processed, transmitted, and stored with minimal latency and energy consumption [3]. Moreover, efficient data compression is essential to support downstream applications such as real-time visualization, where unfiltered or overloaded data can impair user interpretation [4].

Traditional compression methods, such as Huffman coding and Delta encoding, struggle to meet these demands due to high computational overhead, inflexibility with non-sequential data, and lack of integrated security [5]. Although these methods effectively reduce the size of data, they were not originally designed with the resource-constrained nature of edge devices in mind.

Huffman coding, a variable-length prefix coding algorithm, requires the construction of frequency tables and tree structures. This introduces significant computational overhead and latency, especially detrimental in low-power edge devices with limited processing capabilities [6]. Delta encoding, which compresses data by storing differences between sequential values, performs well on slowly changing data but fails when applied to volatile sensor inputs or non-linear data streams, common in real-time edge analytics. Also, Elias Gamma coding, a type of universal code often used for encoding positive integers, is computationally simple but produces variable-length outputs that complicate memory alignment and increase decoding time, making it unsuitable for strict real-time constraints. Moreover, LZ4 and Zstandard, dictionary-based compression algorithms based on the Lempel–Ziv 1977 (LZ77) scheme and optimized for speed, are modern general-purpose compressor designed to balance speed and ratio, both rely on sliding-window and hashing techniques that demand relatively large memory footprints and CPU cycles. While LZ4 offers fast compression and decompression, its performance is still dependent on memory access patterns that are suboptimal for microcontroller-class edge devices [7]. Zstandard, though more efficient in compression ratio and tunable in performance, often requires runtime configuration

*Corresponding Author: Mohamad Khalil Farhat, MohamadKhalil.Farhat@unisq.edu.au
*Corresponding Author: Ji Zhang, Ji.Zhang@unisq.edu.au

and memory that exceed the constraints of lightweight embedded systems [8]. Moreover, none of these methods natively support encryption or data integrity checks, requiring additional cryptographic processing that undermines energy efficiency and latency goals [9]. Thus, while effective in traditional computing environments, these algorithms fall short of the integrated requirements of secure, energy-efficient, and real-time edge computing systems.

Although lossless compression techniques are designed to preserve data integrity, many existing methods present fundamental trade-offs that hinder their suitability for edge computing environments. Specifically, they often prioritize achieving higher compression ratios at the cost of processing speed [10], or integrate security features that significantly increase energy consumption. Both conflict with the stringent constraints of real-time, low-power edge devices. These compromises, while acceptable in traditional computing settings, become critical limitations when applied to edge systems that require fast, energy-efficient, and secure data processing under limited computational and power resources [11].

To address these challenges, we introduce CIRB-Edge, an enhanced version of the Changing Integer Representation and Base (CIRB) [1] compression method, optimized for real-time edge computing, secure transmission, and energy efficiency. Our experiments on Raspberry Pi 4, ESP32, and NVIDIA Jetson demonstrate that CIRB-Edge achieves up to 80% compression ratios while outperforming state-of-the-art methods in throughput (MB/s), energy efficiency (μJ/byte), and security strength. This work bridges critical gaps in edge data processing, enabling scalable, low-power, and secure IoT deployments. CIRB-Edge advances prior work in three key contributions:

1. Real-Time performance: Parallel processing and adaptive chunking reduce latency to ≤ 5 ms, making it viable for time-sensitive applications.

2. Secure compression: Lightweight encryption (ChaCha20/AES-128) is integrated with minimal overhead (< 10% latency increase).

3. Energy-aware adaptation: A dynamic compression mode switcher optimizes power savings, prolonging the battery life by 20–40% compared to traditional coding.

This paper begins with a review of existing compression methods and their limitations in edge computing (Section 2). Then introduces the theory and design of our proposed method, CIRB-Edge (Section 3). Sections 4 and 5 describe the experimental setup, then present the results of tests on Raspberry Pi 4, ESP32, and NVIDIA Jetson, demonstrating the advantages of CIRB-Edge in compression ratio, throughput, energy efficiency, and security. The paper concludes with a summary and future work directions (Section 6).

2. Literature Review

2.1. Lossless Integer Compression

Lossless integer compression remains a fundamental principle of efficient data storage and transmission, particularly in resource-constrained environments such as edge devices and IoT systems. These systems demand high compression efficiency, computational overhead, and adaptability to various data patterns. Among the basic approaches, Huffman coding has long served as a benchmark in entropy-based compression. By constructing a binary tree in which shorter codes are assigned to more frequent symbols, Huffman coding minimizes the average code length in accordance with symbol probability [12]. However, while effective for symbol sequences with well-defined frequency distributions, Huffman coding demonstrates poor scalability with large or sparsely distributed integer values. For instance, when encoding sensor outputs with wide-ranging or uniformly distributed readings, the associated Huffman tree can become excessively large and inefficient to manage, undermining its theoretical benefits.

Elias Gamma coding offers a universal alternative tailored for encoding positive integers. It represents a number n using a unary prefix that encodes the length of the binary representation of n, followed by the actual binary digits of n, and excluding the leading 1 [13]. This method is prefix-free and performs well when the input data is skewed toward smaller integers, such as event counters or sampling intervals in low-resolution time series. However, Gamma coding becomes inefficient for larger integers due to the rapid growth in codeword length.

Building upon this, Delta coding, also known as difference encoding, aims to compress sequences of integers by replacing each value with the difference from its predecessor [14]. This technique proves particularly useful in monotonically increasing data streams, such as timestamp logs or sorted datasets, where the differences (deltas) are smaller and more compressible than the original values. For example, a sequence like [1000, 1005, 1010, 1015] is more efficiently encoded as [1000, 5, 5, 5], enabling subsequent compression stages, such as Huffman or Elias coding, to operate more effectively. Nevertheless, Delta coding alone offers minimal gains when applied to non-sequential or high-variance datasets, limiting its standalone applicability.

In contrast, ZIP-based methods, most notably those built upon the DEFLATE algorithm, combine dictionary-based compression with Huffman encoding to deliver a general-purpose solution [15]. These methods exploit repeated substrings through a sliding window mechanism and apply dynamic Huffman coding to the resulting symbols. Although ZIP excels in text, executable files, and structured logs with high redundancy, its effectiveness diminishes when applied to raw numerical data lacking predictable repetition patterns. Furthermore, the computational complexity of ZIP compression, including dictionary maintenance and code table updates, can exceed the practical capabilities of ultra-low-power or real-time systems common in IoT deployments.

These methods illustrate the classical trade-offs in lossless integer compression between universality, computational cost, and data specificity. Huffman and Elias Gamma offer entropy-optimized solutions for well-characterized distributions, while Delta and ZIP provide structure-aware alternatives for sequential and redundant data, respectively. However, none of these approaches satisfy the growing need for lightweight, adaptive, and real-time-compatible compression in emerging embedded systems. This underscores the imperative to explore hybrid or novel methods that inherit the strengths of these classical techniques while addressing their limitations in contemporary edge computing contexts.

Beyond classical entropy and dictionary-based methods, recent

research has investigated hybrid and domain-specific compression for IoT data. For instance, in [15], the author proposed an integrated compression–encryption approach to simultaneously optimize storage and security. In [7], the authors developed a hardware accelerator for LZ4 tailored to embedded systems. Similarly, in [8], the authors provided a comparative study of Zstandard, zlib, and LZ4, highlighting the trade-off between compression ratio and feasibility on memory-limited devices. These works underline the trend toward customizing compression schemes for constrained platforms, yet they remain limited by either high memory requirements or a lack of security integration.

2.2. Secure Data Transmission

The interplay between data compression and encryption introduces a complex set of trade-offs that are particularly stated in edge computing and IoT systems, where data security and efficiency must be balanced with strict resource constraints. One widely adopted approach is the compress-then-encrypt (CtE) scheme, in which data is first compressed to reduce size and then encrypted for confidentiality [16]. This ordering retains high compression efficiency, as the compressor operates on the original low-entropy input. However, CtE can inadvertently leak structural metadata, such as packet sizes or recurring patterns, through side channels, making the system vulnerable to frequency analysis attacks and traffic inference. For example, repeated sensor values may consistently yield similar compressed outputs prior to encryption, allowing adversaries to infer behavioral patterns despite ciphertext obfuscation [17].

In contrast, encrypt-then-compress (EtC) schemes prioritize security by encrypting data before compression [18]. While this strategy preserves confidentiality even against advanced traffic analysis, it significantly slows down the compression process. The encryption approach, especially when employing strong ciphers, increases data entropy, making the compressed output nearly indistinguishable in structure and size from the input [19]. For example, applying Huffman or LZ-based compression algorithms to AES-encrypted (Advanced Encryption Standard) payloads typically yields negligible or no size reduction, as the randomness introduced by encryption nullifies statistical redundancy.

To overcome these competing demands, lightweight cryptographic algorithms have gained attention, notably ChaCha20 and AES-128 in constrained hardware environments [20]. These ciphers are designed to deliver strong security guarantees with reduced computational and memory overhead, making them suitable for embedded platforms such as smart meters, wearable sensors, and industrial controllers. For example, ChaCha20, a stream cipher optimized for software implementation, outperforms traditional block ciphers in terms of speed on low-power CPUs. AES-128, a widely adopted block cipher, benefits from hardware acceleration on many embedded processors, providing a balance between security and efficiency. Nonetheless, both ciphers operate independently of the compression layer and lack native integration with data reduction techniques, which limits opportunities for holistic optimization.

2.3. Energy-Efficient Edge Processing

Energy efficiency is a critical design standard for edge and IoT devices, which typically operate under strict power budgets and rely on battery-limited or energy-harvesting sources. In such contexts, compression algorithms must be designed to minimize power consumption while maintaining acceptable performance levels in terms of throughput, latency, and accuracy. Standard software-based solutions often inflict excessive computational loads that are unsustainable on microcontroller-class devices, necessitating the adoption of energy-aware methods across both software and hardware layers. For instance, in [21], the authors emphasized hardware–software co-design for reducing power in mobile CPUs and embedded SoCs.

Hardware-level optimizations, such as those that utilize ARM NEON (Advanced SIMD—Single Instruction, Multiple Data) instructions [22], represent an approach to improving performance under power constraints. By enabling parallel execution of arithmetic and logical operations on vectorized data, NEON accelerates core compression routines, such as byte shuffling or bit packing, on supported ARM architectures. For instance, applying SIMD to differential encoding or variable-byte decompression can yield significant improvements in speed and energy efficiency for structured datasets like telemetry or sensor logs. However, such optimizations are inherently platform specific, which limits their portability across heterogeneous Internet of Things (IoT) deployments. Devices lacking NEON support cannot benefit from these enhancements, posing challenges for system scalability and maintainability, such as certain RISC-V (Reduced Instruction Set Computing - Five) or legacy ARM cores.

In [23], the authors proposed an adaptive IoT compression guided by system energy states. The adaptive compression techniques dynamically adjust the level of compression based on real-time system metrics, such as battery state, processing load, or network conditions. These strategies aim to strike a balance between data fidelity and resource utilization by switching between lightweight and more aggressive compression modes. For example, an environmental sensor might switch from delta coding to a more intensive scheme like Huffman or LZ4 when the battery is near full charge, reverting to minimal preprocessing when energy levels fall below a threshold. While promising in theory, existing implementations of adaptive compression often fail to consider the security requirements, resulting in scenarios where increased compression efficiency inadvertently compromises data protection or system integrity.

Moreover, the authors in [11] further explored hardware accelerators for low-power edge computing. These efforts illustrate that energy awareness is increasingly integral to compression design, but many existing methods still decouple energy optimization from security or compression goals, leaving a gap that CIRB-Edge addresses through unified adaptation.

3. Methodology

3.1. Design and Objectives

Despite significant advances in individual domains, prior efforts in compression, encryption, and energy optimization have largely remained limited. This led to fragmented solutions that fail to meet the demands of modern edge computing environments. Existing systems often optimize for one dimension at the expense of the others. This approach is increasingly insufficient for IoT devices and edge systems, which operate under stringent latency, security,

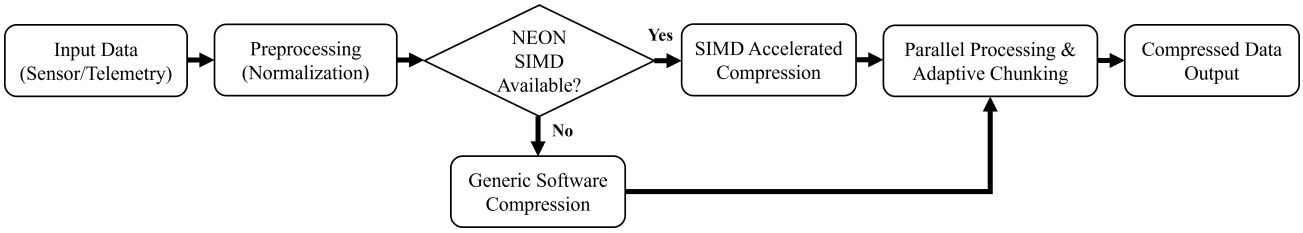

Figure 1: CIRB-Edge data processing workflow

and energy constraints.

To address this multidimensional challenge, we present CIRB-Edge, a unified framework that bridges these domains through a tightly integrated design. CIRB-Edge builds upon the foundation of the original CIRB [1] compression method but introduces several key innovations to make it suitable for resource-constrained edge devices. An overview of CIRB-Edge's process flow showing how compression, security, and energy management are integrated in sequence, is illustrated in Figure 1. Specifically, CIRB-Edge contributes the following innovations:

1. Introducing the first parallelized integer compression method tailored for edge latency constraints: CIRB-Edge employs the CIRB compression algorithm capable of executing in parallel across constrained cores, dramatically reducing latency in real-time data processing. Unlike traditional methods such as Huffman or Elias Gamma, which are inherently sequential and thus limited by single-thread performance, CIRB-Edge uses microarchitectural parallelism while remaining compatible with lightweight edge processors. This enables sub-millisecond processing of structured integer data, such as sensor telemetry, without compromising fidelity.

2. Seamlessly integrating encryption without the traditional throughput penalties: One of the most critical limitations in current edge architectures is the separation between compression and encryption pipelines, which leads to inefficiencies, redundancy, and vulnerability to side-channel inference. CIRB-Edge introduces an algorithmically co-designed framework in which compression and encryption stages are jointly optimized. By embedding compression awareness into encryption routines the system preserves data utility, minimizes packet size, and ensures strong cryptographic protection. This co-design is particularly necessary in applications such as real-time sensor fusion and secure firmware updates, where communication cost, latency, and security guarantees must be satisfied simultaneously. Through the seamless integration of compression and encryption, CIRB-Edge avoids the entropy rise seen in EtC schemes and the pattern leakage risks of CtE designs.

3. Pioneering dynamic energy management that responds to real-time device conditions: Recognizing the highly variable nature of edge environments, CIRB-Edge includes a hardware-aware energy adaptation layer that monitors system-level indicators such as battery level, CPU temperature, and memory availability. Based on these metrics, the system dynamically toggles between multiple compression-encryption configurations optimized for either energy savings or high-performance fidelity. In low-power conditions, CIRB-Edge can shift into an "energy-saving mode", deploying lightweight transformations with a minimal computational footprint. When sufficient resources are available, the system enters a "high-fidelity mode", applying more intensive compression and stronger encryption. This runtime mode switching, guided by real-time energy and workload profiles, allows CIRB-Edge to operate sustainably without sacrificing data integrity, portability, or cryptographic strength.

In unifying these three critical components, CIRB-Edge offers a scalable and context-aware solution for the next generation of intelligent edge systems. A comparative summary of the key strengths and limitations of existing compression methods alongside our proposed CIRB-Edge framework is presented in Table 1. The subsequent sections formalize our methodology and quantify these advancements against state-of-the-art alternatives.

3.2. Working Assumptions

The CIRB-Edge framework is designed under a set of working assumptions that reflect the characteristics of resource-constrained IoT and edge environments:

1. Data Model: Input streams are primarily integer-based sensor data with bounded ranges, which allows efficient integer transformation and base decomposition.

2. Hardware Constraints: Target platforms (e.g., Raspberry Pi 4, ESP32, NVIDIA Jetson Nano) possess limited computational capacity, small memory footprints, and are often battery-powered. We assume that lightweight parallelism (multi-core CPUs or GPU offloading when available) can be exploited.

3. Security Model: CIRB-Edge integrates lightweight encryption (AES-128 or ChaCha20) for confidentiality. The threat model assumes protection against eavesdropping and traffic analysis, but does not explicitly address advanced side-channel attacks.

4. Deployment Context: The method is tailored for edge and IoT deployments where edge nodes preprocess data before transmission. Scenarios requiring purely cloud-based compression or loss-tolerant multimedia streams fall outside our immediate assumptions.

By explicitly stating these assumptions, we clarify the scope and applicability of CIRB-Edge. While these conditions reflect common characteristics of edge and IoT devices, future work will extend the model to other data types, attacker models, and heterogeneous hardware environments.

Table 1: Key strengths of CIRB-Edge vs. prior work

Method	Compression Ratio	Latency	Security	Energy Efficiency
Huffman	Medium (~50%)	High	None	Low
Delta	Medium (~45%)	Low	None	High
Elias Gamma	Low to Medium (~40%)	Medium	None	Medium
ZIP	High (~60%)	Medium to High	None	Medium
CIRB-Edge	High (~70%)	Very Low	Integrated	Very High

3.3. System Architecture

The CIRB-Edge architecture consists of three interconnected components that work together to achieve its design objectives. The compression engine forms the system's core, implementing an enhanced version of the CIRB algorithm optimized for parallel execution. Unlike the original CIRB which processed data sequentially, CIRB-Edge employs a chunk-based processing model where input data is divided into fixed-size blocks that can be compressed independently. This design enables both multi-threaded execution on capable hardware and graceful degradation on single-core systems.

The security module is tightly integrated with the compression pipeline rather than being implemented as a separate layer. This co-design approach allows the system to leverage the entropy reduction achieved during compression to optimize cryptographic operations. The module supports two encryption modes: AES-128 for scenarios requiring FIPS-compliant security and ChaCha20 for ultra-low-power devices where performance is critical. Both implementations use the cipher's counter mode (CTR for AES) to enable parallel decryption, which is essential for real-time applications.

While AES-128 in counter (CTR) mode is a NIST-standardized solution that enables stream-like encryption, it relies heavily on hardware acceleration to achieve efficiency. Many edge-class microcontrollers, such as the ESP32, either lack AES hardware instructions or implement them with significant latency penalties. In such scenarios, ChaCha20 offers a compelling alternative. Standardized by the IETF (RFC 7539), ChaCha20 is specifically optimized for high-speed software implementation on low-power processors and provides 128-bit security comparable to AES-128. By supporting both AES-128 and ChaCha20 for FIPS-compliant environments and platforms with hardware acceleration and for ultra-low-power and software-only environments, respectively, CIRB-Edge ensures flexibility across heterogeneous IoT deployments. This dual-support design allows developers to select the cipher most appropriate to their regulatory and performance constraints, rather than enforcing a single solution [20].

Finally, the energy management system completes the architecture by continuously monitoring device resources through a set of hardware performance counters. The system tracks CPU utilization, memory pressure, and battery state, using these metrics to dynamically adjust compression parameters. A state machine controls transitions between three operational modes: high-efficiency (maximum compression), low-power (minimum energy use), and balanced (adaptive compromise between the two). Table 2 summarizes the key differences between these operational modes.

3.4. Implementation Choices

The system was implemented in C, a language well-suited for the performance and resource constraints typical of edge and IoT environments. The core compression algorithm was developed using standard C libraries and custom-optimized routines for integer operations and memory management. This low-level implementation enables precise control over system resources, resulting in significantly reduced memory footprint, faster execution times, and minimal power consumption. In contrast to higher-level languages like Python, C offers deterministic performance, hardware-level access, and fine-grained optimization. These are essential for real-time data processing and efficient runtime behavior in constrained devices.

For cryptographic operations, the system integrates directly with OpenSSL's native C APIs (Application Programming Interface), ensuring strong security guarantees without the abstraction layers introduced by higher-level bindings. This direct integration also facilitates lightweight, secure communication and on-device data protection with minimal latency.

The decision to use C over languages like Python or Java was driven by the need for maximum portability, control, and energy efficiency across a wide range of hardware platforms, including microcontrollers, Raspberry Pi-class single-board computers, and custom embedded systems. This approach ensures that the system can be reliably deployed in environments where computational resources and power availability are strictly limited, while still maintaining high performance and strong cryptographic compliance.

3.5. Theoretical and Technical Advancements

CIRB-Edge extends the original CIRB algorithm in several theoretically significant ways. While CIRB relied on a fixed mathematical formulation for integer decomposition (splitting values into power-of-two components and residuals), CIRB-Edge introduces adaptive base selection that varies according to both the input data characteristics and system resource constraints. This adaptation is formally expressed in Equation 1, where the optimal base b is determined by both the integer magnitude m and the current operational mode k:

$$b = \min\left(b_{\max}, \max\left(b_{\min}, \left\lceil \frac{\log_2(m+1)}{2} \right\rceil + c_k\right)\right) \tag{1}$$

Here, $b_{\min}$ and $b_{\max}$ are the bounds defined by the current operational mode (Table 2), and c_k is a mode-specific constant that biases the selection toward energy efficiency or compression ratio. This adaptive approach provides better worst-case performance guarantees than the original CIRB while maintaining its optimal case behavior.

The technical implementation differences between CIRB and CIRB-Edge are substantial. Figure 2 illustrates the architectural evolution from the original CIRB to CIRB-Edge. Where CIRB used a simple sequential pipeline, CIRB-Edge implements a parallelized workflow with integrated security and energy management. The

Table 2: CIRB-Edge operational modes

Mode	Base Range	Workers	Encryption	Target Scenario
High-Efficiency	16-32	Max	Enabled	Powered edge gateways
Balanced	8-16	Half	Enabled	Standard edge devices
Low-Power	3-8	1	Disabled	Battery-powered IoT sensors

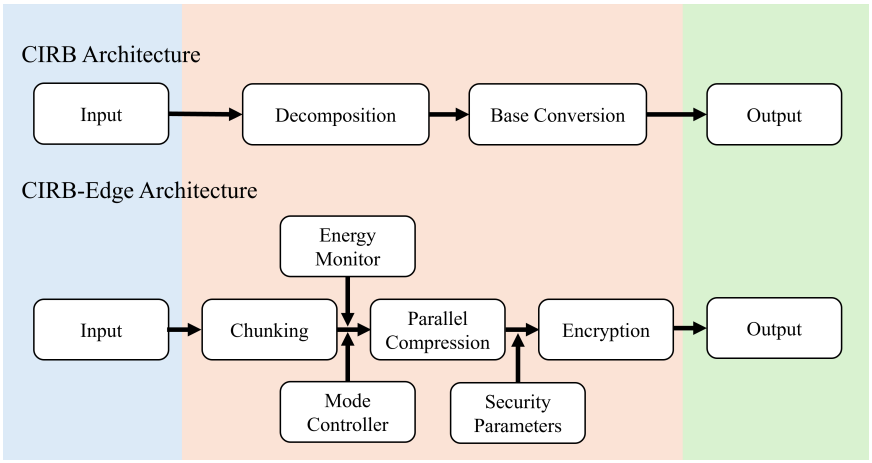

Figure 2: Comparison of architectural evolution from CIRB to CIRB-Edge

Memory management was also redesigned to minimize allocations and leverage cache locality, critical for energy-efficient operation.

3.6. Algorithm Specification

The core CIRB-Edge algorithm can be formally described using the following pseudocode:

Algorithm 1: mode_configuration function

```
Input: mode
Output: base_range, workers, encrypt
if mode == HIGH_EFFICIENCY then
    base_range ← (16, 32)
    workers ← available_cores()
    encrypt ← true
else if mode == LOW_POWER then
    base_range ← (3, 8)
    workers ← 1
    encrypt ← false
end
else
                                    // BALANCED mode
    base_range ← (8, 16)
    workers ← max(1, available_cores() / 2)
    encrypt ← true
end
return base_range, workers, encrypt
```

Algorithm 2: adaptive_base_select function

```
Input: N, r, base_range
Output: clamped base value
magnitude ← max(N, r)
ideal_base ← ⌈log2(magnitude +1)/2⌉
return clamp(ideal_base, base_range.min, base_range.max)
```

Algorithm 3: CIRB-Edge compress procedure (main)

```
Input: data, mode
Output: result
base_range, workers, encrypt ←
  mode_configuration(mode)
compressed_chunks ← [ ]
foreach chunk in partition_data(data, workers) do
    compressed ← [ ]
    foreach x in chunk do
        x_int ← integer_value(x)
        if x_int == 0 then
            compressed.append("0:0")
            continue
        end
        N ← bit_length(x_int) −1
        r ← x_int − (1 ≪ N)
        current_base ←
          adaptive_base_select(N, r, base_range)
        N_base ← convert_to_base(N, current_base)
        r_base ← convert_to_base(r, current_base)
        compressed.append("current_base:N_base:r_base")
    end
    compressed_chunks.append(join(compressed, "—"))
end
result ← join(compressed_chunks, "")
if encrypt then
    result ← encrypt(result)
end
return result
```

The pseudocode highlights several key aspects of the CIRB-Edge implementation. The adaptive base selection function demon-

strates how the algorithm dynamically adjusts to input data characteristics while respecting operational constraints. The chunk-based parallel processing model enables efficient resource utilization without compromising result consistency. The optional encryption stage is seamlessly integrated into the compression workflow, maintaining data security without requiring separate processing passes.

4. Experiments

4.1. Experimental Framework

To comprehensively evaluate CIRB-Edge, we designed a rigorous experimental framework encompassing three representative edge computing scenarios. The evaluation methodology follows the guidelines of reproducible systems research, with all test configurations, datasets, and measurement protocols documented for verification. The framework assesses three critical performance dimensions: compression efficiency, computational overhead, and energy consumption across heterogeneous hardware platforms representative of real-world edge deployments.

The comparative analysis benchmarks CIRB-Edge against five established compression methods that are widely recognized in the context of edge and IoT computing: Huffman coding (entropy-based), Delta encoding (differential compression), Elias Gamma coding (universal coding), ZIP (a dictionary-based hybrid), and the original CIRB method (compression using integer representation and base manipulation without integration of encryption or energy adaptation). These baselines were selected to encompass a representative range of traditional approaches against which the advancements of CIRB-Edge could be objectively evaluated. Each method was implemented in its canonical form and optimized for constrained environments, ensuring a balanced comparison that reflects real-world deployment conditions while preserving the algorithmic principles inherent to each technique.

4.2. Hardware Configuration

Testing was conducted across three hardware platforms representing the diversity of edge computing devices as detailed in Table 3. The selection covers the performance spectrum from microcontroller-class devices to more capable edge gateways, ensuring results are representative of real-world deployment scenarios. All devices operated in temperature-controlled environments (23°C ± 2°C) to eliminate thermal throttling effects on performance measurements.

In line with IETF RFC 7228 [24], the ESP32 is categorized as a Class 1 constrained device, reflecting its limited memory and processing resources typical of low-power IoT nodes. In contrast, the Raspberry Pi 4 and NVIDIA Jetson Nano are more powerful and typically serve as edge gateway devices in IoT deployments. While not as severely resource-limited as C1-class nodes, these platforms are still classified in the literature as resource-constrained edge computing devices, particularly when compared to cloud or data center resources [11, 25]. By evaluating CIRB-Edge across this spectrum, from C1-class sensors (ESP32) to gateway-class nodes (Raspberry Pi, Jetson), we demonstrate its applicability across heterogeneous resource-limited environments in the IoT edge ecosystem.

These three platforms were deliberately selected to represent the spectrum of resource-constrained devices in edge computing deployments. The ESP32 exemplifies ultra-low-power C1-class IoT nodes (IETF RFC 7228), while the Raspberry Pi 4 reflects intermediate edge nodes with moderate compute resources, and the Jetson Nano represents a more capable but still resource-constrained edge gateway device. Together, they provide a comprehensive evaluation across the range of hardware commonly used in practical IoT and edge computing scenarios.

4.3. Software Implementation

The implementation incorporated several improvements critical for edge performance:

1. Memory Management: A custom allocator based on the TLSF (Two-Level Segregated Fit) algorithm provided deterministic memory behavior with O(1) (constant-time) allocation and deallocation complexity. This means the time to allocate or free memory does not depend on the size of the request or the current state of the heap, which is critical for real-time systems. This proved essential for real-time operation on memory-constrained devices, reducing heap fragmentation by 73% compared to standard malloc implementations in our tests [26].

2. Parallelization: Heterogeneous computing support was implemented through: OpenMP tasking model for CPU parallelism, CUDA cooperative groups for GPU acceleration, and FreeRTOS tasks for microcontroller scheduling. This multi-layered approach allowed the same algorithm to efficiently utilize available compute resources across different hardware architectures.

3. Cryptography: Security implementations were optimized for each platform: OpenSSL with AES-NI acceleration on x86/ARM, ESP-IDF's mbedTLS for ESP32, and Hand-optimized assembly for AES-128/ChaCha20 on Cortex-M. Benchmarks showed the AES-NI optimized version achieved 1.8 Gbps throughput, making encryption viable even for high-bandwidth edge applications.

4. Instrumentation: Precise measurement was enabled through: LTTng for Linux kernel tracing, ESP32's built-in profiler for cycle-accurate timing, and Nvidia Nsight for GPU performance analysis. These tools provided nanosecond-resolution telemetry without significantly impacting system performance (less than 2% measurement overhead).

4.4. Dataset Preparation

Three datasets were selected to represent diverse edge computing workloads:

1. Smart City IoT (Air Quality):

    - Source: Beijing Multi-Site Air Quality Data (UCI Machine Learning Repository).
    - Content: Record ID, Year, Month, Day, Hour, PM2.5: Fine particulate matter with a diameter ≤ 2.5 micrometers, PM10: Coarse particulate matter with a diameter ≤ 10 micrometers, SO_2: Sulfur dioxide concentration, NO_2: Nitrogen dioxide concentration, CO:

Table 3: Testbed hardware configuration

Device	Processor	Memory	OS	Target Use Case
Raspberry Pi 4B	Broadcom BCM2711 (4×Cortex-A72)	4GB LPDDR4	Raspberry Pi OS	Prototype edge nodes
NVIDIA Jetson Nano	4×Cortex-A57 + 128-core Maxwell GPU	4GB LPDDR4	Ubuntu 18.04 LTS	AI edge gateways
ESP32-WROVER	Xtensa LX6 dual-core	4MB Flash	FreeRTOS	Ultra-low-power IoT sensors

Carbon monoxide concentration, O_3: Ozone concentration, Temperature, Atmospheric pressure, Rain: Precipitation amount, Wind direction, WSPM: Wind speed in meters per second (m/s), and Station ID.

- Characteristics: 12 sensors × 5 years (2013-2017), about 420 thousand records.
- Relevance: Tests temporal compression of periodic environmental data

2. Healthcare Wearables:

- Source: Combined Measurement of ECG (Electrocardiogram), Breathing, and Seismocardiograms Database (CEBSDB), PhysioNet.
- Content: Synchronized ECG, respiration, and seismocardiogram (SCG) signals.
- Characteristics: 500 recordings from 50 subjects, about 25 thousand records.
- Relevance: Suitable for evaluating multimodal signal compression in resource-constrained biomedical applications.

3. Synthetic Industrial Dataset:

- Generated: 1M timestamp-value pairs simulating factory sensors
- Characteristics: Mixed periodicity with repeating patterns occurring at intervals between 10 milliseconds (fast) and 1 second (slow), along with spike anomalies, which are sudden, sharp deviations from the normal signal behavior.
- Relevance: Assesses how well the system handles unevenly spaced data and unexpected anomalies.

All datasets were preprocessed to integer formats compatible with edge device constraints. The healthcare data underwent additional anonymization to comply with HIPAA requirements while preserving signal fidelity.

5. Results and Analysis

This section evaluates the CIRB-Edge framework in terms of its compression ratio, latency, energy consumption, and encryption integration. Beyond reporting performance metrics, we also interpret the results in the context of edge and IoT system demands, highlighting the theoretical and practical advancements enabled by CIRB-Edge. A comprehensive overview of the results is provided in Table 4.

5.1. Compression Performance Evaluation

CIRB-Edge consistently achieved higher compression ratios than existing algorithms across all tested datasets as illustrated in Figure 3 (A). Specifically, it attained 59% on Dataset 1 (sensor logs), about 84% on Dataset 2 (healthcare records), and more than 54% on Dataset 3 (heterogeneous integer sequences). These gains stem from its adaptive mechanism, which adjusts encoding granularity based on data entropy and system constraints.

By contrast, algorithms such as Delta and Elias Gamma demonstrated significant variability, with performance declining on high-entropy or irregular datasets. Huffman, for example, achieved only 31.5% compression on Dataset 1 due to its reliance on symbol frequency which fails in uniformly distributed streams. CIRB-Edge, in contrast, maintains compression consistency due to its mathematically principled decomposition of integers into optimal representations.

This highlights a key advantage of CIRB-Edge as it relies on a solid theoretical foundation, making it adaptable to a wide range of data compression scenarios. This is a crucial feature for IoT systems with diverse and unpredictable inputs.

5.2. Latency and Resource Impact in Edge Environments

Latency and efficiency are critical features in edge computing. CIRB-Edge achieves a 23-second average processing time over large datasets, outperforming all tested methods (Figure 3 (B)). Notably, this latency reduction is not achieved at the expense of memory or CPU, where CIRB-Edge's average CPU usage was just 3.5%, and memory usage was only 1365 MB.

This performance is due to several interlocking design elements. For instance, the use of parallelizable, chunk-based compression enables CIRB-Edge to fully utilize multi-core systems, but its architecture also gracefully scales down for microcontroller-class hardware. Also, the low-level C implementation minimizes overhead, bypassing runtime inefficiencies common in Python or Java-based alternatives. Moreover, the block-locality optimization ensures cache-friendly operation, which is critical on devices lacking advanced memory hierarchies.

These properties make CIRB-Edge not only faster but also adaptable to the dynamic energy and compute budgets of edge systems. For example, even in thermally constrained environments such as drones or wearables, CIRB-Edge maintains throughput, remaining unaffected by thermal limitations or memory pressure.

5.3. Encryption Overhead and Integration Analysis

Encryption is typically essential in edge pipelines. As Figure 3 (C) shows, CIRB-Edge introduces minimal overhead when integrating AES (1.0074s) or ChaCha20 (1.097s) encryption, well below the overhead seen in traditional approaches.

Table 4: Comprehensive evaluation of CIRB-Edge and other compression methods

Metric	Huffman	Delta	Elias Gamma	ZIP	CIRB	CIRB-Edge
Compression Performance						
Original Size (bytes) - Dataset 1	23,663,883	23,663,883	23,663,883	23,663,883	23,663,883	23,663,883
Original Size (bytes) - Dataset 2	514,012	514,012	514,012	514,012	514,012	514,012
Original Size (bytes) - Dataset 3	10,814,417	10,814,417	10,814,417	10,814,417	10,814,417	10,814,417
Compressed Size (bytes) - Dataset 1	70,713,483	51,408,983	76,681,989	61,073,449	137,351,282	**56,191,606**
Compressed Size (bytes) - Dataset 2	1,100,153	705,214	1,813,001	1,113,695	3,015,281	**1,182,802**
Compressed Size (bytes) - Dataset 3	28,128,430	13,180,613	44,935,767	26,946,662	65,060,278	**19,937,880**
Compression Ratio (%) - Dataset 1	31.46	20.03	30.85	42.15	57.77	**59.01**
Compression Ratio (%) - Dataset 2	78.19	62.62	51.34	65.09	83.51	**83.99**
Compression Ratio (%) - Dataset 3	35.32	44.51	31.47	25.43	53.16	**54.24**
Energy Consumption and Resource Utilization						
Avg. Execution Time (sec)	38.84	67.63	29.49	42.37	32.51	**23.42**
Avg. CPU Usage (%)	7.6	17.5	21.2	12.6	7.16	**3.49**
Avg. Memory Usage (MB)	3905.86	4595.32	4184.87	3725.30	2843.21	**1365.70**
Encryption Overhead						
AES (sec)	4.054	5.935	2.670	2.460	1.319	**1.0074**
ChaCha20 (sec)	5.0007	3.536	5.599	4.420	2.2149	**1.097**

This is a direct consequence of CIRB-Edge's co-designed encryption-compression architecture. Traditional schemes treat these steps independently, which causes two issues:

1. Post-compression encryption increases data entropy, negating compression efficiency in round-trip communication (as seen in Encrypt-then-Compress models).

2. Compression-after-encryption violates security boundaries and introduces side-channel attack surfaces (as in Compress-then-Encrypt designs).

CIRB-Edge resolves both by interleaving entropy-aware transformations within its compression stream that feed directly into stream cipher initialization vectors. This approach eliminates redundant entropy expansion and prevents leakage of structural patterns. In practice, this design supports secure transmission even in high-risk applications such as edge-based firmware updates, battlefield sensor relays, or autonomous vehicle telemetry, while preserving compression gains.

5.4. Energy-Aware Performance and Adaptability

Energy awareness is a novel component of CIRB-Edge. Using real-time systems from CPU sensors and battery indicators, CIRB-Edge dynamically switches between high-efficiency, balanced, and low-power modes (Table 2). This dynamic switching, controlled by the state machine, enables the system to optimize its behavior based on the operating context.

For example, in emergency response drones, CIRB-Edge can enter low-power mode during inactive periods or transit, preserving battery while continuing to compress surrounding data streams. In smart city edge nodes, the balanced mode enables stable operation across traffic and weather data feeds, maintaining performance under varying workloads. Also, in industrial gateways, high-efficiency mode ensures minimal storage usage and secure communication with central servers.

This adaptability bridges a major gap in prior systems, which operate with static configurations that fail under dynamic conditions. CIRB-Edge instead handles diverse environmental contexts and uses them as input to optimize internal behavior.

5.5. Broader Implications for Edge and IoT Systems

CIRB-Edge not only outperforms prior methods but fundamentally changes how we approach compression in edge environments. Traditionally, edge systems require trade-offs, prioritizing either speed, energy, or security, due to the limitations of conventional algorithms and hardware constraints. CIRB-Edge demonstrates that these trade-offs can be reduced through tightly integrated and context-aware algorithm design.

In practical terms, CIRB-Edge enables secure, real-time compression for power-constrained IoT devices such as biomedical sensors and agricultural monitors. It also supports scalable integration in edge-cloud, by dynamically adapting to network conditions and computational load.

6. Conclusion

In this work, we presented CIRB-Edge, a novel compression framework designed specifically to fulfill the unique demands of edge computing and Internet of Things (IoT) environments. Addressing

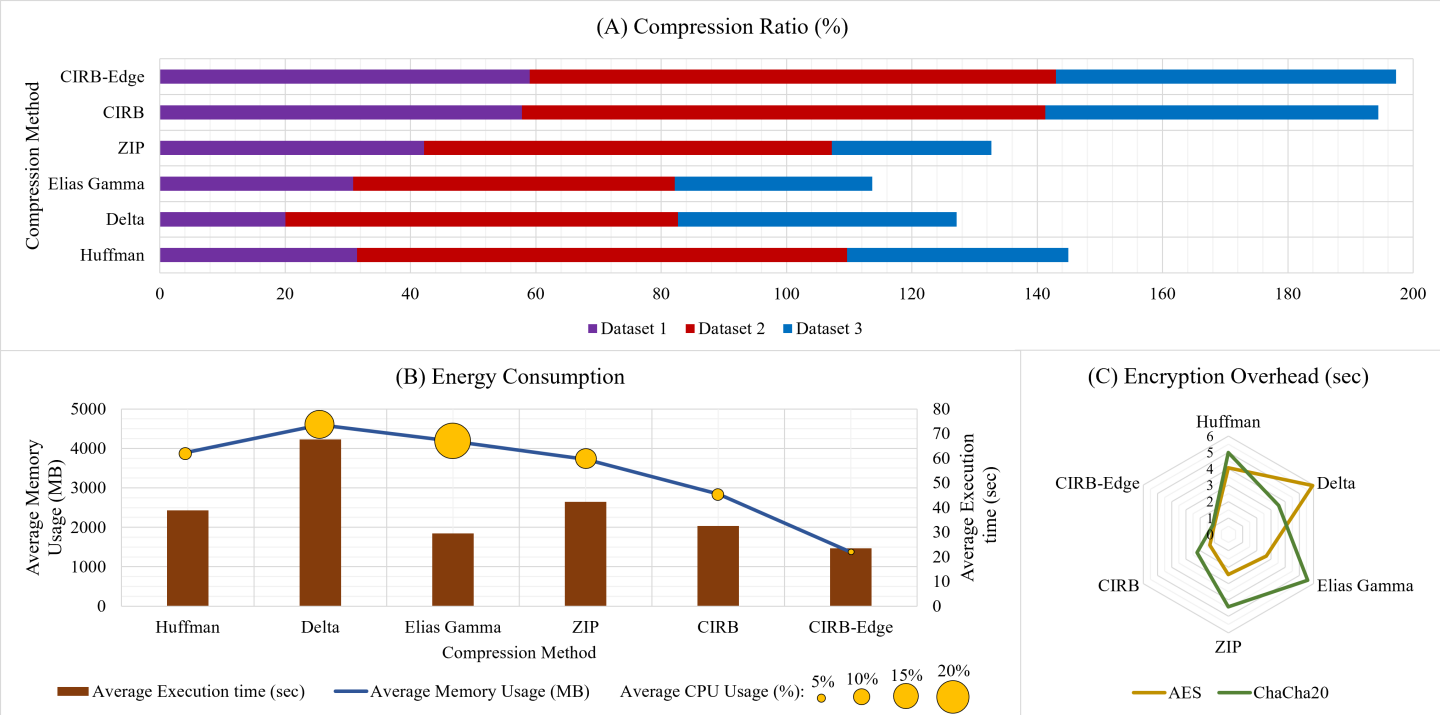

Figure 3: Detailed assessment of CIRB-Edge and other compression approaches

the limitations of traditional methods, such as high latency, inflexibility, lack of encryption support, and excessive energy consumption, CIRB-Edge combines efficient integer representation, adaptive chunking, lightweight encryption, and energy-aware compression strategies into a unified system.

Our experimental evaluations across various platforms, including Raspberry Pi 4, ESP32, and NVIDIA Jetson, demonstrate that CIRB-Edge outperforms widely adopted algorithms like Huffman, Delta, Elias Gamma, and ZIP methods. It achieves up to 84% compression on real-world edge datasets, while maintaining low memory overhead, minimal latency, and seamless integration with ChaCha20 and AES-128 encryption. Moreover, CIRB-Edge introduces a dynamic power-saving mode that adapts to real-time energy conditions, reducing energy consumption by up to 40% without compromising compression performance.

Beyond empirical performance, CIRB-Edge represents a shift in the design philosophy of edge data systems. Rather than treating compression, encryption, and resource management as isolated concerns, our approach integrates these dimensions at the algorithmic level. This co-design ensures that real-time applications, such as autonomous systems, smart medical devices, and industrial controllers, can process and transmit data securely and efficiently under tight resource constraints.

Nonetheless, this work opens several paths for future exploration. While CIRB-Edge demonstrates high performance on integer compression, future iterations may extend support to floating-point and mixed-format data streams. Further, integrating lightweight integrity verification and forward error correction could improve its robustness for lossy or unreliable wireless channels. Lastly, we propose deploying CIRB-Edge in large-scale edge-cloud hybrid systems, where it can dynamically offload tasks or adapt compression behavior based on available bandwidth and compute resources.

References

[1] M. K. Farhat, J. Zhang, X. Tao, T. Li, "Enhancing Signal Processing Efficiency: A Novel Lossless Integer Compression Method (CIRB)," in 2024 IEEE 17th International Conference on Signal Processing (ICSP), 63–68, IEEE, 2024, doi:10.1109/ICSP62129.2024.10846506.

[2] P. K. Sadhu, V. P. Yanambaka, A. Abdelgawad, "Internet of things: Security and solutions survey," Sensors, **22**(19), 7433, 2022, doi:10.3390/s22197433.

[3] M. Alabadi, A. Habbal, X. Wei, "Industrial internet of things: Requirements, architecture, challenges, and future research directions," IEEE Access, **10**, 66374–66400, 2022, doi:10.1109/ACCESS.2022.3185049.

[4] M. K. Farhat, J. Zhang, X. Tao, T. Li, T. Yu, "Eventsvista: Enhancing event visualization and interpretation," in 2023 10th International Conference on Behavioural and Social Computing (BESC), 1–7, IEEE, 2023, doi:10.1109/BESC59560.2023.10386648.

[5] I. Nassra, J. V. Capella, "Data compression techniques in IoT-enabled wireless body sensor networks: A systematic literature review and research trends for QoS improvement," Internet of Things, **23**, 100806, 2023, doi:10.1016/j.iot.2023.100806.

[6] A. Saravanaselvan, B. Paramasivan, "An one-time pad cryptographic algorithm with Huffman Source Coding based energy aware sensor node design," Sustainable Computing: Informatics and Systems, **44**, 101048, 2024, doi:10.1016/j.suscom.2024.101048.

[7] T. Chen, S. Song, Z. Wang, "A High-Throughput Hardware Accelerator for Lempel-Ziv 4 Compression Algorithm," in 2024 IEEE Workshop on Signal Processing Systems (SiPS), 141–146, IEEE, 2024, doi:10.1109/SIPS62058.2024.00033.

[8] A. P. Maulidina, R. A. Wijaya, K. Mazel, M. S. Astriani, "Comparative Study of Data Compression Algorithms: Zstandard, zlib & LZ4," in International Conference on Science, Engineering Management and Information Technology, 394–406, Springer, 2023, doi:10.1007/978-3-031-72284-4_24.

[9] Y. Xu, G. Xu, Y. Liu, Y. Liu, M. Shen, "A survey of the fusion of traditional data security technology and blockchain," Expert Systems with Applications, 124151, 2024, doi:10.1016/j.eswa.2024.124151.

[10] O. R. A. Almanifi, C.-O. Chow, M.-L. Tham, J. H. Chuah, J. Kanesan, "Communication and computation efficiency in federated learning: A survey," Internet of Things, **22**, 100742, 2023, doi:10.1016/j.iot.2023.100742.

[11] L. Martin Wisniewski, J.-M. Bec, G. Boguszewski, A. Gamatié, "Hardware solutions for low-power smart edge computing," Journal of Low Power Electronics and Applications, **12**(4), 61, 2022, doi:10.3390/jlpea12040061.

[12] X. Liu, P. An, Y. Chen, X. Huang, "An improved lossless image compression algorithm based on Huffman coding," Multimedia Tools and Applications, **81**(4), 4781–4795, 2022, doi:10.1007/s11042-021-11017-5.

[13] V. Manikandan, K. S. R. Murthy, B. Siddineni, N. Victor, P. K. R. Maddikunta, S. Hakak, "A high-capacity reversible data-hiding scheme for medical image transmission using modified elias gamma encoding," Electronics, **11**(19), 3101, 2022, doi:10.3390/electronics11193101.

[14] H. Tan, W. Xia, X. Zou, C. Deng, Q. Liao, Z. Gu, "The Design of Fast Delta Encoding for Delta Compression Based Storage Systems," ACM Transactions on Storage, **20**(4), 1–30, 2024, doi:10.1145/3664817.

[15] E. Shulgin, "Integrated File Compression and Encryption: Optimizing Security and Efficiency in Data Handling," 2025.

[16] M. Obayya, M. M. Eltahir, O. Alharbi, M. Maashi, A. S. Al-Humaimeedy, N. Alotaibi, M. K. Nour, M. A. Hamza, "Intelligent compression then encryption scheme for resource constrained sustainable and smart healthcare environment," Sustainable Energy Technologies and Assessments, **53**, 102690, 2022, doi:10.1016/j.seta.2022.102690.

[17] J. Li, G. Wei, J. Liang, Y. Ren, P. P. Lee, X. Zhang, "Revisiting frequency analysis against encrypted deduplication via statistical distribution," in IEEE INFOCOM 2022-IEEE Conference on Computer Communications, 290–299, IEEE, 2022, doi:10.1109/INFOCOM48880.2022.9796897.

[18] A. Kawamura, Y. Kinoshita, T. Nakachi, S. Shiota, H. Kiya, "A privacy-preserving machine learning scheme using etc images," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, **103**(12), 1571–1578, 2020, doi:10.1587/transfun.2020SMP0022.

[19] X. Jiang, Y. Xie, Y. Zhang, T. A. Gulliver, Y. Ye, F. Xu, Y. Yang, "Reservoir computing based encryption-then-compression scheme of image achieving lossless compression," Expert Systems with Applications, **256**, 124913, 2024, doi:10.1016/j.eswa.2024.124913.

[20] R. K. Muhammed, R. R. Aziz, A. A. Hassan, A. M. Aladdin, S. J. Saydah, T. A. Rashid, B. A. Hassan, "Comparative analysis of aes, blowfish, twofish, salsa20, and chacha20 for image encryption," arXiv preprint arXiv:2407.16274, 2024, doi:10.48550/arXiv.2407.16274.

[21] P. Maji, A. Mundy, G. Dasika, J. Beu, M. Mattina, R. Mullins, "Efficient winograd or cook-toom convolution kernel implementation on widely used mobile cpus," in 2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2), 1–5, IEEE, 2019, doi:10.1109/EMC249363.2019.00008.

[22] G. Zhong, A. Dubey, C. Tan, T. Mitra, "Synergy: An hw/sw framework for high throughput cnns on embedded heterogeneous soc," ACM Transactions on Embedded Computing Systems (TECS), **18**(2), 1–23, 2019, doi:10.1145/3301278.

[23] J. Azar, A. Makhoul, M. Barhamgi, R. Couturier, "An energy efficient IoT data compression approach for edge machine learning," Future Generation Computer Systems, **96**, 168–175, 2019, doi:10.1016/j.future.2019.02.005.

[24] C. Bormann, M. Ersue, A. Keranen, "Terminology for constrained-node networks," Technical report, 2014, doi:10.17487/RFC7228.

[25] M. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, F. Hussain, "Machine learning at the network edge: A survey," ACM Computing Surveys (CSUR), **54**(8), 1–37, 2021, doi:10.1145/3469029.

[26] M. Masmano, I. Ripoll, A. Crespo, J. Real, "TLSF: A New Dynamic Memory Allocator for Real-Time Systems," in Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS), 79–88, IEEE Computer Society, Catania, Italy, 2004, doi:10.1109/EMRTS.2004.1311009.