STUDY AND DEVELOPMENT OF EFFICIENT DATA COMPRESSION TECHNIQUES

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

In
Electronics and Communication Engineering

by
MUKESH SAHU
(2K18/PHD/EC/11)

Under the Supervision of

Prof. J. Panda

Department of Electronics and Communication Engineering

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, main Bawana Road, Delhi-110042, India

November, 2025

DEPARTMENT OF ELECTRONICS AND COMMUNICTION ENGG.
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042.

CANDIDATE'S DECLARATION

I, **MUKESH SAHU**, Roll No. **2K18/PHD/EC/11** hereby certify that the research work which is being presented in the thesis entitled **"STUDY AND DEVELOPMENT OF EFFICIENT DATA COMPRESSION TECHNIQUES"** in partial fulfilment of the requirements for the award of Doctor of Philosophy, submitted in the Department of **Electronics and Communication Engineering**, Delhi Technological University, Delhi, is an authentic record of my own work carried out during the period from 1st August,2018 to August,2025 under the supervision of **Prof. Jeebananda Panda**. The thesis embodies result of original work, and studies are carried out by the student himself.

The matter presented in this thesis has not been submitted by me elsewhere in part or fully to any other University or Institute for the award of any degree.

Date: MUKESH SAHU

Place:

DEPARTMENT OF ELECTRONICS AND COMMUNICTION ENGG.
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042.

CERTIFICATE BY THE SUPERVISOR

This is to certify that the research work which is being presented in the thesis, titled **"STUDY AND DEVELOPMENT OF EFFICIENT DATA COMPRESSION TECHNIQUES"** by **Mukesh Sahu**, Roll No. **2K18/PHD/EC/11** as a part-time scholar in the Department of Electronics and Communication Engineering, Delhi Technological University, in fulfilment of requirements for the award of Doctor of Philosophy is an authentic work carried by him under my supervision. The thesis embodies result of original work, and studies are carried out by the student himself. The matter presented in this thesis has not been submitted elsewhere in part or fully to any other University or Institute for the award of any degree to the best of my knowledge and belief.

Date:

Supervisor
Prof. Jeebananda Panda
Professor
Department of ECE
Delhi Technological University,
Delhi –110042

DEPARTMENT OF ELECTRONICS AND COMMUNICTION ENGG.
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042.

ACKNOWLEDGEMENT

First and foremost, I give my deepest thanks to Almighty God for granting me the strength, wisdom, and opportunities to undertake and successfully complete this research. Without His blessings, none of this would have been possible.

I wish to express my profound gratitude to Prof. Jeebananda Panda sir from the Department of Electronics and Communication Engineering, Delhi Technological University, for his unparalleled guidance, dedication, and unwavering support throughout this journey. His academic prowess, coupled with his invaluable mentorship, has been instrumental in shaping this research. His exceptional ability to steer me through the complexities of this work has truly been a source of inspiration, motivating me to persevere despite the many challenges. It is under his mentorship that I have grown both as a researcher and as an individual, and for that, I am forever grateful.

I would also like to extend my sincere appreciation to the Head of the Department, Prof. Neeta Pandey madam, for her consistent encouragement and for fostering a conducive academic environment. I am equally thankful to all the esteemed faculty members of the Department of Electronics and Communication Engineering, Delhi Technological University, for their insightful lectures, constructive feedback, and for providing a platform that encouraged collaborative learning. Their academic excellence and dedication to research have always served as a beacon of inspiration for me.

A special note of thanks goes to my family for their unconditional love, prayers, and ceaseless encouragement. My parents, my wife Renu, my kids Uday and Abhay in particular, have been a constant source of inspiration, offering me the emotional strength to stay focused and committed to my goals.

Mukesh Sahu
2K18/PHD/EC/11
PHD Scholar
Department of Electronics and Communication Engineering,
E-mail: mukeshsahu@ieee.org

Study and Development of Efficient Data Compression Techniques

Mukesh Sahu

ABSTRACT

The explosive rise of digital data across healthcare, environmental monitoring, industrial automation, and smart cities has intensified the need for efficient and reliable data compression. Resource-constrained platforms such as wireless sensor networks (WSNs) and Internet of Things (IoT) devices face the sharpest challenges: they operate with limited memory, energy, and processing power, yet must transmit critical information without error. Conventional compression methods are not always suited to these constraints, and because the data cannot tolerate distortion, lossless compression is essential. The central aim of this thesis is to study existing techniques and to develop new methods of lossless compression that are faster, lighter, and more adaptive to practical environments.

The work unfolds in six major phases, each addressing a distinct gap. Phase 1 begins with a comprehensive benchmarking study of classical algorithms such as Huffman, RLE, LZW, Zstd, LZMA, and LZ4 across synthetic data, real sensor traces, and Indic-language corpora. This baseline analysis revealed clear limitations in speed, memory demand, and energy efficiency when applied to low-power devices. Building on this, Phase 2 introduces a syntactic approach, the LZWP algorithm, which improves throughput by refining dictionary updates while maintaining compression ratio, making it suitable for real-time IoT workloads.

In Phase 3, the focus shifts to semantic compression through ontology-driven frameworks. An ontology-based healthcare pipeline and OntoRLE for WSNs demonstrated that lightweight semantic preprocessing could uncover hidden structure in data, leading to better compressibility without compromising accuracy. Phase 4 explores hybrid pipelines, combining algorithms such as cascade of Zstd and LZ4HC, then applying them to Devanagari Hindi corpora and IoT text streams. These cascaded methods achieved a stronger balance between compression ratio, speed, and decompression efficiency compared to standalone techniques.

Phase 5 presents one of the thesis's most significant contributions: MOR-ALDC (Memory-Optimised Residual Adaptive Lossless Data Compression). By combining residual transformation with canonical Huffman coding, MOR-ALDC reduced memory requirements and improved both compression efficiency and execution speed. This design was particularly effective for WSNs, where memory footprint and energy usage are critical bottlenecks.

The final stage, Phase 6, brings these advances closer to deployment through Artificial Intelligence guided models and hardware acceleration. A decision module was integrated into an FPGA-based compressor to determine, in real time, whether compression should be applied and which algorithm is best suited under given energy and bandwidth conditions. This edge framework, validated through prototype experiments, confirmed that intelligent, adaptive compression can operate reliably at scale in IoT systems while saving energy and improving responsiveness.

Across all phases, the proposed methods consistently outperformed classical baselines in compression ratio, memory footprint, throughput, and energy savings. These results carry significant implications for real-world applications. In environmental monitoring, the methods enable long-lived sensor deployments in remote or sensitive areas. In healthcare, they allow reliable, low-cost transmission of patient data in telemedicine settings. In digital inclusion, they improve efficiency for Indic-language text, supporting local-language applications and e-governance. And in IoT and edge computing, the Artificial Intelligence guided framework contributes to greener, more sustainable digital infrastructure.

The research also opens clear directions for future work. These include on-node continual learning to adapt decisions as data evolves, dynamic orchestration of hybrid pipelines, and semi-automatic ontology generation to reduce human effort in new domains. Further work on cross-sensor compression, ASIC-level implementations, lightweight security integration, and long-term field trials will strengthen both technical maturity and real-world reliability.

Overall, the thesis progresses systematically from benchmarking to syntactic, semantic, hybrid, and statistical approaches, before advancing into Artificial

Intelligence driven hardware acceleration. Each stage responds to specific gaps, while together they form a coherent framework for efficient, adaptable, and deployable lossless compression. Beyond technical contributions, the research highlights broader social impact by supporting sustainable sensor networks, affordable healthcare, inclusive digital services, and more reliable critical systems.

Research Publications of
Mukesh Sahu (2K18/PHD/EC/11).

List of Research Papers

S. No	**Title**	**Conference / Journal**	**DOI**	**Publication Date**
1.	A Time Efficient Approach to Data Compression for LZW Algorithm.	AICERA 2023 (IEEE Conference)	10.1109/AICERA/ICIS59538.2023.10420326	November, 2023
2.	A Proposed IoT-based Smart Healthcare Management Framework using Concept of Ontology.	JSIR (Vol. 83, Issue 3) (SCIE)	10.56042/jsir.v83i3.6153	March, 2024
3.	OntoRLE: An Ontological-Based Compression Algorithm for Improving Energy Efficiency and Memory Utilization.	Int. J. of Information Technology (Springer)	10.1007/s41870-024-02277-z	November, 2024
4.	An IoT-Based Edge Computing Lossless Compression Approach for Enhancing Energy Efficiency in Networks.	JSIR (Vol. 84, No. 6) (SCIE)	10.56042/jsir.v84i6.11252	June, 2025
5.	MOR-ALDC: A Memory-Optimised Residual Adaptive Lossless Compression Algorithm for Energy-Efficient WSNs. **(Won the Best Paper Award.)**	ICDAM, 2025. Scopus (International Conference.)	DOI: https://doi.org/10.1007/978-3-032-03769-5_38	June, 2025 (Presented)
6.	Hybrid Lossless Compression for Devanagari-Encoded Hindi Text: Performance Evaluation and Methodological Advancements.	Sadhna Journal, Indian Academy of Sciences.	_	Communicated in April, 2025.
7.	AIoT-Enabled FPGA Accelerator for Adaptive, Low-Power Signal Compression in Wireless Sensor Networks.	PreMI,2025, IIT Delhi.	_	Communicated in July, 2025.

TABLE OF CONTENTS

List of Tables

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE

A. Symbols

Symbol	Description / Formula	Unit
b	Number of transmitted bits	bits
CP	$\mathrm{CP} = \frac{\mathrm{Size}_{\mathrm{original}} - \mathrm{Size}_{\mathrm{compressed}}}{\mathrm{Size}_{\mathrm{original}}} \times 100$	%
CR	$\mathrm{CR} = \frac{\mathrm{Size}_{\mathrm{original}}}{\mathrm{Size}_{\mathrm{compressed}}}$	–
CS	$\mathrm{CS} = \frac{\mathrm{Size}_{\mathrm{original}}}{\mathrm{Compression\ Time}}$	MB/s
CT	Compression Time – total time required to complete compression	Seconds (s)
d	Transmission distance between nodes	meters (m)
DS	$\mathrm{DS} = \frac{\mathrm{Size}_{\mathrm{compressed}}}{\mathrm{Decompression\ Time}}$	MB/s
DT	Decompression Time – time required to reconstruct original data	Seconds (s)
E	Total Energy Consumption = Power × Time (≈ 0.15 W × T)	mJ
$\mathrm{E}_{\mathrm{Compressed}}$	Energy required to transmit compressed data	mJ
$\mathrm{E}_{\mathrm{elec}}$	Electronic energy consumed per bit	J/bit
$\mathrm{E}_{\mathrm{Original}}$	Energy required to transmit original data	mJ
E_{tx}	$\mathrm{E}_{\mathrm{tx}} = \mathrm{b} \times \mathrm{E}_{\mathrm{elec}} + \mathrm{b} \times \varepsilon_{\mathrm{amp}} \times$	Joules (J)
$\varepsilon_{\mathrm{amp}}$	Energy used by the transmitter amplifier per bit per m²	J/bit·m²

ES%	$ES\% = \frac{E_{Original} - E_{Compressed}}{E_{Original}} \times 100$	%
MAX	Maximum possible signal value (e.g., 255 for 8-bit data)	–
N	Number of packets or samples used for averaging	–
NL	Network Lifetime – time until the first sensor node exhausts energy	Rounds / Seconds
PWSR	Peak Working-Set RAM – maximum memory allocated during compression	KB
PSNR	$PSNR = 10\log_{10}\left(\frac{MAX}{MSE}\right)$	dB
RMSE	$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{S}_i - S_i)^2}$	Bytes
RSSΔ	Resident-Set Size Delta – increase in total RAM usage during execution	KB
$Size_{compressed}$	Size of the compressed data	Bytes
$Size_{original}$	Size of the original (uncompressed) data	Bytes
SP	Stability Period – rounds completed before first node failure	Rounds
T	Runtime duration used in the energy model	Seconds (s)
S_i	Original packet size of the i-th sample	Bytes
$\hat{S}_i$	Reconstructed packet size after decompression	Bytes

B. Abbreviations

Abbreviation	Full Form
AI	Artificial Intelligence
AIoT–MOR-ALDC	Artificial Intelligence of Things – Memory-Optimised Residual Adaptive Lossless Data Compression.
ALDC	Adaptive Lossless Data Compression
BWT	Burrows–Wheeler Transform
CCSDS	Consultative Committee for Space Data Systems
DCT	Discrete Cosine Transform
DWT	Discrete Wavelet Transform
FPGA	Field Programmable Gate Array
GIF	Graphics Interchange Format
IoT	Internet of Things
JPEG	Joint Photographic Experts Group
LZ	Lempel–Ziv
LZ4 / LZ4HC	Lempel–Ziv 4 / High Compression variant

Abbreviation	Full Form
LZMA	Lempel–Ziv Markov Chain Algorithm
LZW	Lempel–Ziv–Welch
ML	Machine Learning
MOR-ALDC	Memory-Optimised Residual Adaptive Lossless Data Compression
RAM	Random Access Memory
RLE	Run-Length Encoding
WSN	Wireless Sensor Network
Zstd	Zstandard Compression

C. Nomenclature (Key Terms and Definitions)

Term	Description
Lossless Compression	Compression method that allows perfect data reconstruction after decompression.
Residual Transformation	Encoding of differences between predicted and actual data values to reduce redundancy.

Term	Description
Canonical Huffman Coding	Memory-efficient variant of Huffman coding with lexicographically ordered codewords.
Hybrid Codec	A compression setup combining two algorithms sequentially (e.g., LZW + Huffman).
Semantic Preprocessing	Pre-analysis of data to exploit structure or meaning before compression.
Energy-Aware Compression	Compression optimized for minimum energy use in IoT/WSN devices.
Ontology-Driven Compression	Data encoding guided by semantic relationships among sensor attributes.
FPGA Implementation	Hardware realization of compression algorithms for high speed and energy efficiency.
Edge-Level Compression	Local data reduction at the sensor or IoT node before transmission.

CHAPTER 1

INTRODUCTION

The exponential growth of digital technologies in recent decades has led to an unprecedented increase in the amount of data being generated every second. Applications ranging from healthcare and industrial automation to environmental monitoring and smart cities rely on the continuous flow of data for decision-making and efficient operation. Managing this vast volume of information requires techniques that not only reduce storage needs but also allow faster and more reliable communication. Among these, efficient data compression techniques has emerged as an essential tool because it reduces data size while preserving accuracy. This chapter introduces the motivation for the study, outlines the key concepts of data compression, and presents the problem context, research objectives, scope, significance, and overall organization of the thesis.

1.1 Background and Motivation

Although lossless compression has been studied extensively, most established methods were designed for environments with abundant memory and processing power. With the emergence of wireless sensor networks (WSNs) and low-power IoT devices, the requirements have shifted dramatically. These platforms typically operate with only a few kilobytes of memory, limited computational ability, and strict energy budgets, yet they must process diverse and continuous data streams in real time.

Traditional algorithms such as Huffman coding, LZW, and hybrid pipelines deliver competitive compression ratios but often demand significant memory or computational resources. This makes them less suitable for embedded or battery-powered systems, where efficiency and adaptability are as important as compression effectiveness. Furthermore, real-world applications increasingly require algorithms that can adjust to varied data types and still remain lightweight enough for practical deployment.This gap between conventional compression approaches and the unique constraints of IoT and WSN platforms defines the motivation for the present research. The central aim is to develop and evaluate techniques that are not only efficient in reducing data size but are also adaptive, resource-aware, and feasible for real-world deployment on constrained devices.

1.2 Fundamentals of Data Compression

Data compression is the process of transforming original data into a more compact representation by identifying and eliminating redundancies or patterns. Its primary objective is to reduce the size of data while either preserving its full integrity (in the case of lossless compression) or maintaining acceptable quality (in lossy compression). The main motivation behind compression is to reduce storage space, accelerate data transmission, and enhance overall system efficiency.

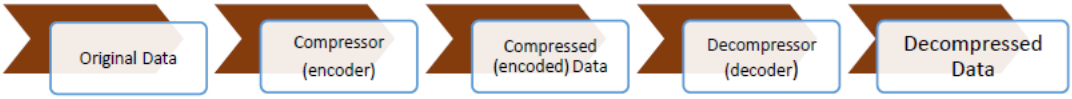

Fig. 1.1: General Data Compression And Decompression Process

The general workflow of data compression and decompression is illustrated in Fig. 1.1, which shows how the original data is passed through a compressor (encoder), transmitted or stored in compressed form, and later reconstructed by a decompressor (decoder) at the receiving end. In general, data compression involves two stages:

i. **Modeling**: In this step, the data is analyzed to identify repeating patterns or statistically frequent symbols. A model is constructed to represent the data more compactly.
ii. **Coding**: Here, the data is encoded using fewer bits based on the model. Often, symbols that appear more frequently are assigned shorter codes, and infrequent ones are assigned longer codes.

1.2.1 Types of Data Compression

Data compression algorithms are broadly classified into two categories:

i. **Lossless Compression**: This ensures the exact reconstruction of the original data after decompression. It is essential in domains where accuracy is critical, such as medical imaging, scientific data, financial records, and program files. Popular lossless techniques include Run-Length Encoding (RLE), Huffman coding, Arithmetic Coding, and Lempel-Ziv variants like LZ77, LZ78, and LZW.

ii. **Lossy Compression:** In this case, some data is discarded during compression. It is typically used for images, audio, and video where exact replication is not required, and minor losses are imperceptible to human senses. Lossy methods achieve higher compression ratios through quantization and approximation techniques.

Lossy and lossless techniques are not mutually exclusive. In fact, many lossy algorithms employ lossless compression as a final stage to encode the quantized data, thus benefiting from the advantages of both approaches. This thesis, however, concentrates exclusively on lossless compression, since IoT and WSN applications demand exact data recovery.

1.3 Problem Context and Need for Research

Despite many advances in the field, existing lossless compression techniques show notable shortcomings when deployed in real-world IoT and Wireless Sensor Network (WSN) environments.some are summarized below:

i. **Lack of adaptability:** Most conventional algorithms treat all data in the same way, ignoring its structure or semantics. This reduces their effectiveness when dealing with varied inputs such as sensor readings, biomedical signals, or multilingual text.

ii. **High resource demands:** Methods that achieve good compression ratios often do so at the cost of higher memory usage or slower execution. This is unsuitable for devices working with only a few kilobytes of RAM and tight energy budgets.

iii. **Rigid hybrid approaches:** While multi-stage compression pipelines can be powerful, they are usually fixed in design and do not adapt well across different data types or hardware platforms.

iv. **Narrow evaluation focus:** Much of the existing research emphasizes compression ratio alone. In practice, other factors such as energy efficiency, latency, and hardware feasibility are equally important, particularly in embedded or battery-operated systems.

v. **Variety of modern datasets:** Applications today involve everything from periodic environmental logs and healthcare telemetry to large-scale text databases, demanding solutions that are flexible and robust across domains.

These issues underline the need for compression techniques that are not only efficient in terms of data size reduction but also lightweight, adaptive, and feasible for deployment on constrained devices. This forms the practical and technical basis for the work presented in this thesis.

1.4 Objectives of the Research

The present research is guided by the following objectives:

i. To develop new preprocessing algorithms to improve dataset performance.

ii. To develop an architecture that can be adapted to yield greater throughput.

iii. To develop an efficient data compression algorithm for memory organization in IoT and WSN nodes.

iv. To develop new compression/decompression algorithms for next-generation communication systems.

v. To develop energy-efficient compression techniques using FPGAs.

1.5 Research Scope and Coverage

This thesis is devoted to the study and developmentof efficient lossless data compression techniques, particularly for Internet of Things (IoT) and Wireless Sensor Network (WSN) environments where devices often operate with very limited memory, energy, and processing capacity. While compression ratio has traditionally been the central focus, this work also emphasizes energy efficiency, processing speed, memory usage, and hardware feasibility, since these are equally critical for real-world deployment.

The research scope covers several dimensions. One focus is on enhancing classical methods such as LZW and Huffman coding through lightweight adaptations. Examples include residual transformation combined with canonical Huffman coding to achieve high efficiency at low memory cost, and frequency-pruned LZW variants designed to reduce computational burden without loss in performance. Another direction is semantic and domain-specific compression, where structural patterns inherent in the data (e.g., healthcare telemetry, environmental monitoring logs) are exploited through preprocessing and representation techniques to yield higher compression effectiveness.

The thesis also investigates hybrid pipelines that combine two algorithms in sequence, such as Zstd followed by LZ4HC, which are tested for applications in multilingual text (including Devanagari-encoded Hindi) and sensor data. These approaches are evaluated for their adaptability to diverse datasets. Building on these algorithmic advances, the work introduces AI-assisted decision modules that can guide compression workflows in resource constrained nodes. Lightweight models for signal classification, anomaly detection, and energy-aware decision-making are incorporated to dynamically determine whether to compress, bypass, or adaptively encode data. This demonstrates how modern AI techniques can be integrated with traditional compression to enhance efficiency in real-world IoT deployments.

Validation is carried out on diverse datasets including synthetic data, environmental sensor streams, biomedical monitoring signals, and multilingual text ensuring that the results are broadly applicable.Finally, the thesis explores hardware feasibility by implementing selected algorithms and modules on FPGA platforms and low-power edge devices. RTL simulation and synthesis analysis confirm compliance with LUT and BRAM budgets, demonstrating that the proposed methods are not only effective in software but also practical for hardware deployment.

The scope of this thesis remains within lossless compression techniques, with lossy methods discussed only for comparative purposes. By bridging algorithmic design, AI-based adaptability, and hardware validation, the work provides deployable solutions that are both efficient and scalable for next-generation IoT and WSN systems.

1.6 Significance of the Study

Efficient data compression is essential not only for minimizing storage requirements but also for enabling faster, more cost-effective, and scalable data transmission across diverse platforms. While several compression methods exist, practical deployment often demands a balance between compression efficiency, processing speed, memory footprint, and energy consumption, a balance that many current approaches fail to achieve.

Through this research we addresses that challenge by advancing lossless compression techniques that are both performance oriented and deployment-ready. The contributions extend beyond improving compression ratio to include runtime efficiency, resource awareness, and hardware feasibility.The techniques and insights developed here are relevant to a broad spectrum of domains, including:

i. Large-scale data archiving in digital libraries, repositories, and enterprise storage.

ii. Real-time data streams from scientific experiments, industrial monitoring systems, and healthcare instrumentation.

iii. Text processing and storage, with a particular emphasis on multilingual and non-Latin scripts such as Devanagari.

iv. Embedded and constrained systems where memory, energy, and processing budgets are tightly limited.

By combining algorithmic innovations, hybrid pipelines, and hardware level validation, this study creates a direct pathway from research concepts to practical implementation. The outcomes hold the potential to reduce operational costs, increase system longevity, and open new possibilities for efficient data management in both centralized and distributed environments.

1.7 Organization of the Thesis

The thesis is organized into nine chapters, each focusing on a specific dimension of the research work:

1. Chapter 2 – Literature Review presents a survey of classical and modern data compression methods, covering dictionary-based algorithms, entropy coders, hybrid pipelines, and semantic approaches, while highlighting their strengths and limitations. Research Gaps consolidates insights from existing work and formulates the research gaps guide this study.

2. Chapter 3 – Methodology explains the experimental framework, datasets, performance metrics, and implementation tools adopted throughout the research.

3. Chapter 4 – Baseline Analysis of Classical Techniques benchmarks standard algorithms such as Huffman, LZW, RLE, LZMA, Zstd, and LZ4 on diverse datasets, establishing baseline performance for comparison with advanced methods.

4. Chapter 5 – Time Efficient Dictionary Based Compression (LZWP). Presents an optimized LZW variant (LZWP) with frequency-pruned initialization, designed to reduce encoding time while retaining compression efficiency. Includes implementation details, examples, and benchmarking results demonstrating its advantage in resource-limited scenarios.

5. Chapter 6 – Details ontology-driven approaches for healthcare and WSN data. Includes (i) an ontology-based healthcare framework that normalizes patient records before compression, and (ii) the OntoRLE algorithm for WSNs that uses semantic

clustering. Results are compared against baselines using CR, throughput, PSNR/RMSE, stability period, and network lifetime.

6. Chapter 7 – Hybrid Lossless Compression Techniques investigates hybrid codec cascades applied to Devanagari Hindi text data. Benchmarks compression ratio, speed, decompression efficiency, and overall compression effectiveness. The most efficient hybrid algorithm was identified from all the possible combinations of the selected algorithms. Discusses trade-offs and suitability for real-time systems

7. Chapter 8 – Residual Based Compression describes algorithmic advances including residual-plus-canonical Huffman setup, demonstrating gains in compression efficiency, speed, and especially low memory and power usage.

8. Chapter 9 – Divided in two parts first is Edge and and other is FPGA Acceleration with AI-Guided Compression focuses on system-level deployments. It presents a three-tier IoT–Edge–Cloud framework and AIoT–MOR-ALDC on a hardware-accelerated compressor core on FPGA, enhanced with an AI-based decision module that dynamically determines when and how to compress based on signal characteristics and energy conditions. The feasibility of these techniques is validated in terms of real-time performance and hardware resource utilization.

9. Chapter 10 – Conclusions , Future Work and Social Impact summarizes the overall findings, highlights the thesis contributions, and suggests future directions such as adaptive retraining, multi-sensor data fusion, explainable AI, and advanced FPGA/ASIC implementations.Social impact of the findings of the work done in the thesis is also discussed.

1.8 Concluding Remark

This chapter set the stage for the thesis by outlining the motivation, background, fundamentals, objectives, and scope of the study. It emphasized the importance of developing efficient compression techniques that are lightweight and adaptable to the constraints of IoT and WSN environments. Building on this foundation, the next chapter reviews existing approaches, highlights their limitations, and draws out the research gaps that shape the direction of the work.

CHAPTER 2

LITERATURE REVIEW

This chapter reviews how lossless data compression evolved, beginning from its theoretical foundations and walking through the classical algorithms that demonstrated practical feasibility. After this the standardized codecs that enabled broad adoption across industries are studied, then moved to more recent approaches developed for domain-specific and resource-constrained environments. Here, each phase reflects shifting priorities whether the focus was on achieving maximum compression, minimizing computational cost, or ensuring feasibility on real-world hardware. Following this path helped us to show how the field has advanced by balancing efficiency with practicality, and it also reveal the limitations that continue to motivate current research.

A number of different studies provide a valuable context enabling better understanding of this development. Mishra and Singh (2016) gave a broad review covering key categories of entropy-based methods used for compression, dictionary-based techniques, and transform based approaches. Fitriya et al. (2017) extended the perspective by grouping algorithms according to the application domains, and emphasizing mostly on the trade-offs between compression ratio and computational complexity. In contrast, Jayasankar et al. (2021) approached this subject from several dimensions in one go, including data quality, coding schemes, data type, and application area, thereby adding slight difference to earlier reviews. More recently, Ebraheem and Ali (2021) emphasised the importance of aligning algorithm selection with specific constraints, and as per the requirements of the system in which it will be deployed.

Taken together, all these surveys underline two key points: first, nonexistence of a single best method for all contexts, and secondly, the fact that compression research has consistently been shaped by the environment applied. They also point to organizing the literature by data quality (lossless versus lossy), by the type of data (text, image, audio, video), by coding scheme (entropy, dictionary, transform-based), and by the application domain (such as medical imaging or wireless sensor networks). This multi-perspective framework provides the basis for a more detailed discussion of classical, standardised, and modern codecs which is done in the subsequent sections.

2.1 Classical Codecs: Theoretical Foundations and Early Algorithms

The origin of data compression techniques can be traced back to Claude Shannon's pioneering work that he did on information theory (Shannon, 1948). By introducing the concept of entropy, He established a theoretical lower bound on the number of bits required to represent information without loss in any digital system. This benchmark became the foundation for following research, creating a path for the development of algorithms that sought to approach this limit. Holtz (1993) expanded on these ideas by analyzing entropy measures, and early coding schemes, including Huffman and Lempel–Ziv methods, as the tools to quantify redundancy. Some, Broader surveys such as Hosseini (2012) have since provided a comparative evaluation of these algorithms, reinforcing the fundamental role across diverse application domains. Expanding on the classical information and theoretical principles, Devetak and Winter (2003) explored the data compression in the presence of a quantum side information. Their study generalized the concept of conditional entropy to a specific area of quantum systems, establishing limits on compression efficiency when an auxiliary quantum information is available. Although theoretical, this work also reinforces the universalness of entropy-based coding frameworks, that underpin both classical as well as an emerging quantum compression paradigm.

The first most widely adopted practical compression algorithm was Huffman coding (1952). It assigned shorter codes to frequent symbols and longer codes to the infrequent ones, producing an efficient variable-length representation of data. Due to its simplicity and speed, Huffman coding was very quickly integrated into text and image compression systems. Over time, refinements were done. Canonical Huffman coding, for example, organises codewords in lexicographic order, thus enabling the decoder to reconstruct the codebook from code lengths alone, thereby reducing the storage requirements. This variant, adopted in various standards such as JPEG and DEFLATE, as given by Deutsch (1996), was particularly attractive for memory-limited environments like embedded systems and sensor networks.

Golomb coding (1966), he presented an entropy-based compression method which was specifically designed for geometrically distributed data. The technique encodes integers using a quotient and remainder representation, thus achieving an effective balance between coding efficiency and its implementation simplicity. A simplified variant, known as Rice coding, was later proposed by Rice (1979). His method became very particularly popular in space and scientific data compression applications, because of its less computational complexity and suitability for implementations on hardware.

Parallel to all these developments, the Lempel-Ziv (LZ) family introduced dictionary-based compression algorithms. LZ77 (1977) incorporated a sliding window scheme to reference repeated sequences, while LZ78 (1978) built a dictionary of phrases during every encoding. Welch's (1984) refinement led to a new

algorithm widely known as LZW, which had become the basis of popular formats such as GIF and ZIP. All these algorithms demonstrated the practicality of dictionary-based approaches and still continue to influence and motivate modern codecs.

A major breakthrough came when arithmetic coding was introduced by Rissanen (1983), which was further refined by Witten et al. (1987). In this technique entire message is represented as a fractional value within the interval [0, 1]. It is altogether different from the Huffman's symbol-by-symbol encoding approach, In arithmetic coding complete probability distribution of a sequence is modelled, thus achieving compression ratios that approach well within Shannon's theoretical limit. Comparative studies by Porwal et al. (2013) highlighted various advantages of entropy-based methods such as Huffman and arithmetic coding, while Kavitha (2016) showed how different classical approaches RLE, LZW, Huffman, and transform-based methods, each balanced efficiency along with simplicity. Similarly, Azeez and Lasis (2016) again stated the importance of classical algorithms as baselines for all modern work.

Simpler compression approaches have also been playing a significant role in the evolution of lossless data compression. Run-Length Encoding (RLE), which was introduced in the 1950s, compresses data by representing consecutive occurrences of identical values as pairs of symbol and count. Although the efficiency is limited to datasets which contain long sequences of repeated symbols, such as in fax or binary images, RLE demonstrated why it was necessary to adapt compression algorithms to specific data characteristics. In recent years, Flergolla and Wolf (2021) proposed preprocessing-based refinements which have shown an improved RLE's performance on structured datasets, while Klein et al. (2019) have introduced optimized Huffman variants that has been able to achieve compression results closer to the theoretical entropy limit.

Fig 2.1 illustrates the workflows of major classical codecs, Huffman coding, RLE, Lempel–Ziv, arithmetic coding, and Golomb coding. These diagrams complement the theoretical discussion by showing a step-by-step process such as frequency analysis, pattern recognition, code table construction, and compressed output generation.

While all these codecs laid the foundation of the field, they were designed as general-purpose tools and these did not account for the constraints of modern embedded platforms. Their limitations in memory, energy efficiency, and adaptability have led to newer generations of standardized and domain-specific methods. Fig 2.2 provides a broad classification of the lossless compression techniques, dividing all of them into dictionary-based schemes and entropy coding approaches. Together, all these classical methods represent the first systematic demonstrations of redundancy reduction and serve as the enduring baselines for subsequent advancements in data compression.

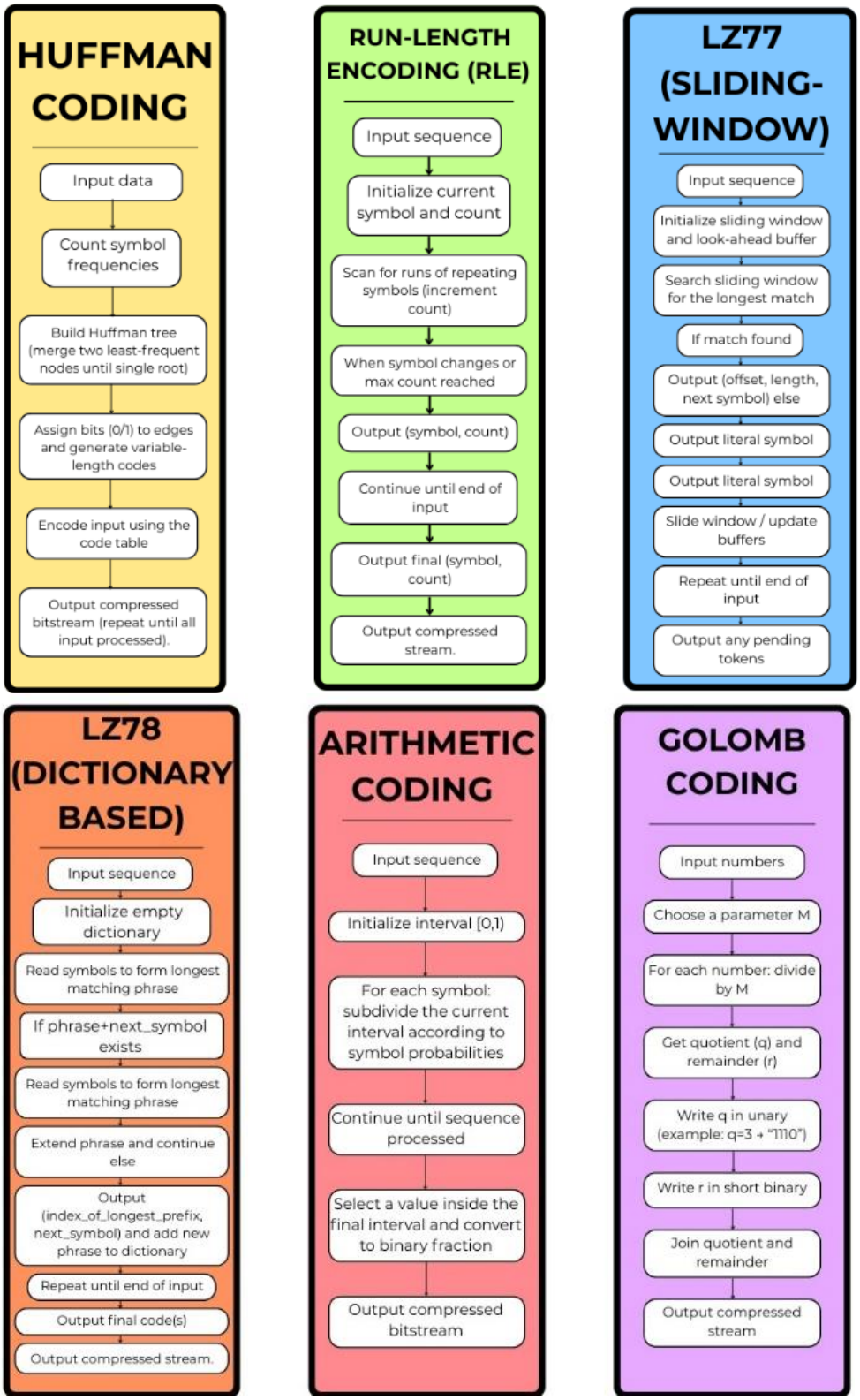

Fig. 2.1: Overview of Fundamental Lossless Compression Algorithms and Their Encoding Steps

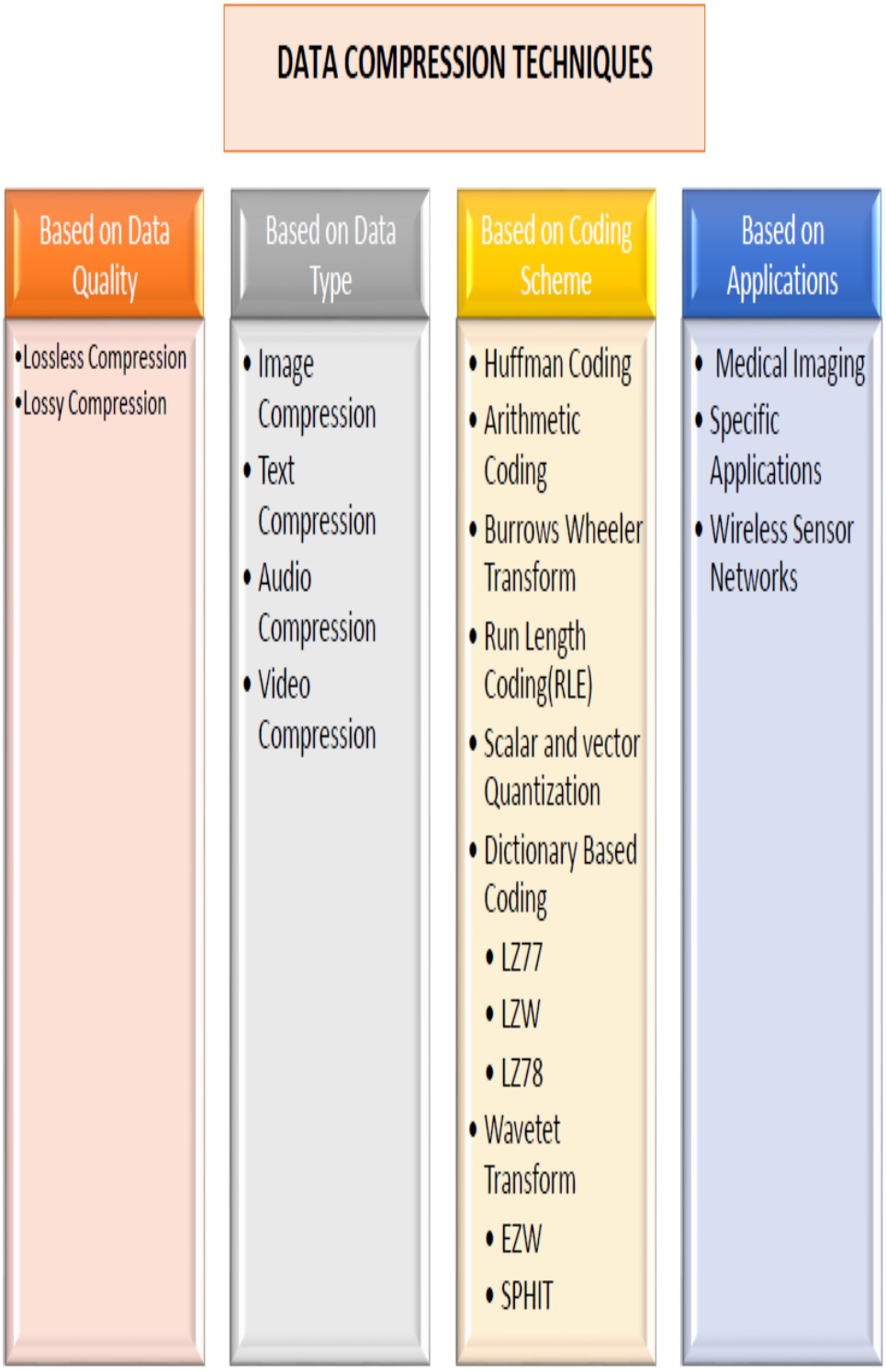

Fig. 2.2: Classification Of Data Compression Techniques by Quality, Type, Coding Scheme, And Applications.

Table 2.1: Comparison of Classical Lossless Compression Algorithms with Their Operating Principles, Advantages, and Limitations

Algorithm	Method Type	Principle	Strengths	Limitations	Applications
Huffman Coding	Entropy-based	Variable-length codes assigned based on symbol frequency	Simple, fast, widely used	Less efficient for sources with skewed distributions	Text compression, image formats (JPEG baseline), DEFLATE
Run-Length Encoding (RLE)	Residual /Transform-based	Encodes consecutive runs of the same symbol as (symbol, count) pairs	Extremely simple, fast	Poor compression for non-repetitive data	Fax transmission, bitmap images, simple sensor data
LZ77 / LZ78	Dictionary-based	Uses sliding window or dictionary to replace repeated patterns with references	Universal, forms basis for many modern codecs	Dictionary maintenance overhead	ZIP, GIF, PNG, Unix compress; CCSDS standards
Arithmetic Coding	Entropy-based	Represents entire message as a sub-interval of [0,1) based on symbol probabilities	Achieves compression close to entropy limit	More computationally complex	Multimedia coding standards, JPEG2000, video coding

A comparative summary of the classical lossless compression algorithms is provided in Table 2.1. The table highlights the various principles, strengths, limitations, and applications of the major codecs which are discussed in this section. However, despite their elegance and efficiency, these classical codecs were primarily designed for the general-purpose algorithms and did not evolve into widely accepted standards. They lack interoperability, robustness, and application-specific tailoring that later generations of standardized codecs have introduced to support industrial and domain-specific needs.

2.2 Standard Codecs: Industry Adoption and Implementations

While the classical codecs have established the theoretical groundwork for the data compression, the widespread adoption of these depended on the development of standardized formats which could guarantee their interoperability across various platforms and different applications. Despite their strong theoretical foundations, early compression works have often overlooked real world constraints including memory usage, encoding time, and compatibility with the streaming data. Addressing

this, Scaglione and Servetto (2002) have proposed models for integrating compression into the multi-hop sensor networks. Their work has demonstrated how compressing correlated sensor readings before routing can lead to significant gains in energy efficiency also improvising performance critical factors for resource-constrained systems like WSNs and IoT nodes.

Standard codecs refer to compression methods that are formalised by international or domain-specific bodies, making them reliable and suitable for use in industries ranging from web communication, image storage to satellite telemetry. Unlike the purely academic algorithms, these codecs were designed with both compression efficiency and robustness in mind, compatibility, and ease of deployment on real hardware systems was also accounted for. Comprehensive surveys by Sayood (2012) and Salomon and Motta (2006) provided structured overviews of how classical schemes such as Huffman coding, arithmetic coding, and the Lempel-Ziv family evolved into practical standards. The analyses of these highlight the transition from theoretical efficiency to real-world implementations, showing us, how industrial adoption was often driven by striking a balance between compression ratio, computational complexity, and hardware feasibility in one go.

This section reviews some of the very most influential standardised codecs. Including widely used formats such as PNG, GIF, and JPEG2000, as well as domain-specific recommendations like CCSDS for remote sensing application. Together, they illustrate how standardisation has bridged the gap between research-oriented algorithms and their deployment in everyday systems for better application.

2.2.1 Image and File Compression Standards

The development of all standardized image compression techniques began with early work on digital facsimile transmission. Hunter and Robinson (1980) have proposed one of the first international standards for digital facsimile coding, establishing a method to efficiently encode and transmit scanned document data. Their contribution provided a conceptual foundation which was used later for image compression standards, where bit-level efficiency and transmission reliability had become central design goals.

One of the earliest and most widely adopted codecs derived from these principles was Graphics Interchange Format (GIF), introduced by CompuServe (1987). GIF employed LZW algorithm to compress bitmap images. Although it was limited to an 8-bit colour depth, its simplicity and due to its broad software support led to its rapid adoption and long-lasting popularity.

In parallel, the demand for higher visual quality in multimedia applications has encouraged the development of wavelet-based standards. JPEG2000, introduced

in the early 2000s, adopted advanced wavelet transforms combined with coding techniques such as Embedded Zerotree Wavelet (EZW) and Set Partitioning in Hierarchical Trees (SPIHT). These methods have enabled superior image quality and higher compression ratios compared to the conventional classical DCT-based JPEG standard. As noted by Taubman and Marcellin (2002), the success of JPEG2000 lies in the ability of it to combine classical concepts with modern transform techniques, resulting in a very flexible and satisfactorily scalable compression system.

2.2.2 Remote Sensing and Space Data Standards

In specialized application domains such as satellite telemetry and remote sensing, the Consultative Committee for Space Data Systems (CCSDS, 2005), established a family of data compression standards, which were specifically tailored to suit for space mission applications. These standards were designed such as to achieve an optimal balance between the compression efficiency, hardware simplicity, and robustness of the system to transmission errors, making them particularly suitable for the resource-constrained on-board systems. They commonly based on predictive modelling and utilised variants of Huffman coding; both were carefully optimised to exploit the statistical characteristics of the scientific and sensor data. A detailed survey by Blanes et al. (2014) reviewed different compression approaches for spaceborne and Earth-observation imaging, and covered standards such as CCSDS-123, CCSDS-122, and JPEG2000. Their work highlighted on how spectral correlation in multispectral and hyperspectral data can be effectively exploited, through prediction or transform-based methods, to achieve significant gains in compression efficiency. Blunier, Magli, and Serra-Sagristà (2014) further examined the optical space imaging systems, emphasizing on how predictive and transform-based methods are adapted to meet the computational limits of the on-board satellite hardware. These studies collectively demonstrated the fact that how standards like CCSDS evolve to balance efficiency, robustness, and implementation feasibility in space environments.

Beyond satellite imaging, the compression standards have also shown significant improvement and have given good results for mesh and 3D geometry data, particularly for use in computer-aided design (CAD) and virtual reality-based applications. Li et al. (2014) reviewed these standards, showing how they enable efficient storage, and proved beneficial on the transmission of complex 3D structures while maintaining geometric integrity.

2.2.3 Wireless Sensor Network Implementations

For the resource-constrained environments such as Wireless Sensor Networks (WSNs), standard codecs must be adapted to reduce both computation as well as the communication overheads. Sadler and Martonosi (2006) proposed Static-LZW (S-LZW) for use in TinyOS motes, they demonstrated substantial energy savings

compared to transmitting uncompressed data. Similarly, Kimura and Latifi (2005) have examined lightweight node compression schemes, emphasizing simplicity and efficiency as critical factors for real-world deployment.

These works highlight the fact that, by combining algorithmic optimizations with hardware-aware design choices, we can get meaningful improvements in speed and energy consumption. Such approaches have strengthened the requirement for embedding compression directly into sensor node operations, rather than treating it as a post-processing step.

2.2.4 Hardware Acceleration of Standard Codecs

The growing demand for real-time data processing, has forced the attention of researchers to implement conventional compression algorithms on dedicated hardware platforms, such as Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs). Hardware implementations enable codecs to meet performance and energy targets, that are often difficult to achieve with software alone. Xu and Zhou (2010), for example, developed an image-compression intellectual-property (IP) core based on a processor and local bus architecture, they demonstrated that FPGA deployment can achieve genuine real-time operation. Similarly, Shah and Vithlani (2011) realised a two-dimensional discrete wavelet transform (2D-DWT) accelerator which utilised distributed arithmetic, enabling efficient image compression with a very modest hardware.

To enhance the performance of dictionary-based codecs, Simrandeep and Sulochana (2012) designed a 5-bit Content Addressable Memory (CAM), which utilises a dictionary structure for the LZW compression algorithm. Their approach has shown to reduce search latency, and improvised data throughput, highlighting the benefits of memory-optimized hardware for accelerating classical algorithms. Building on this, Kamble and Patil (2015) implemented the LZW algorithm on an FPGA platform, showing that how a parallel dictionary operation can significantly improve throughput, and still maintain a lossless performance. These studies have very well illustrated how well-established algorithms like LZW can also be re-engineered for hardware, to meet the demands of embedded and real-time systems.

At the entropy-coding level, Shao et al. (2022) have proposed a VLSI architecture giving high-throughput for a canonical Huffman encoder, it also achieves substantial improvements in speed, and hardware efficiency over traditional designs. Their work has demonstrated a continuous need in the evolution of classical compression methods, through low-level architectural innovation and circuit optimization.

Extending the same line of research, Battini et al. (2014) has introduced a secure image-compression architecture, which was built around a two-dimensional discrete cosine transform (2D-DCT). Their Verilog-HDL implementation on an FPGA, has achieved high throughput, while embedding encryption directly within the compression pipeline. This integration of performance and security clearly explains the potential of hardware algorithm co-designed for next-generation embedded systems.

Collectively, these studies demonstrate how hardware acceleration extends the practical reach of standard codecs, far beyond software-only implementations. FPGA and ASIC platforms consistently deliver notable gains in throughput, latency, and energy efficiency, even though often at the expense of flexibility and design simplicity. Despite this progress, traditional codecs such as PNG, JPEG 2000, and CCSDS remain computationally intensive, and less suited for ultra-constrained platforms like wireless sensor nodes, where both memory as well as power are limited.

A complementary way to classify compression technologies is to do it by its application domain, rather than its coding principle. As illustrated in Fig 2.3, compression strategies primarily align with three major domains: medical imaging, scientific and engineering data, and wireless sensor networks. Each domain imposes unique priorities medical applications emphasise diagnostic fidelity, and resilience to error, whereas WSNs prioritise low power operation and minimal memory usage. This application-driven perspective highlights, how the shared theoretical foundations have evolved, to address diverse real-world requirements. That, naturally leads into the following discussion on domain-specific and hybrid codec designs.

2.2.5 Concluding Remarks on Standard Codecs

Standard codecs represent the foundation on which the compression research moved from theory to practice. By embedding various classical algorithms into formalised formats and specifications, these codecs demonstrated interoperability, reproducibility, and large-scale adoption across various different industries. General-purpose formats such as PNG and ZIP enabled universal file sharing and storage, whereas specialized standards like CCSDS addressed the unique requirements of space missions, and some lightweight adaptations of these were designed for wireless sensor networks (WSNs). Together, these efforts illustrate the steps in which compression methods evolved from academic theories into reliable, industry-ready solutions.

An important aspect of these are the lessons learned from standardisation efforts. They clearly demonstrate that for real-world adoption, we require more than just high compression ratios. We must focus on robustness, compatibility, hardware feasibility, and adaptability. These insights have paved the way for modern codecs, which increasingly integrate hybrid pipelines, domain-specific customization, and, more recently, machine learning approaches. The next section builds on this foundation, by examining contemporary and emerging codecs. These codecs extend beyond standardization to address the demands of specialized and resource-constrained environments.

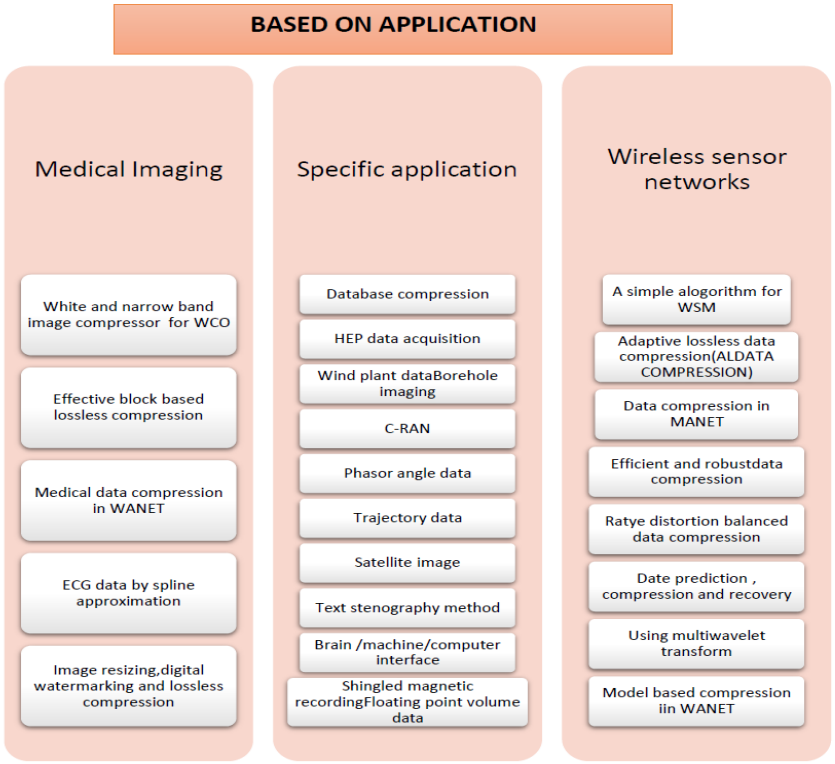

Fig 2.3: Application-Based Classification of Data Compression Techniques

2.3 Modern and Emerging Codecs: Domain-Specific and Hybrid Approaches

Modern data compression research has shown to progress far beyond conventional, algorithm-specific approaches toward a more integrated and adaptive codec designs. These newer frameworks have combined multiple principles such as transform analysis, residual modeling, semantic representation, and hybrid encoding to achieve higher efficiency across a plethora of diverse data types. They are particularly valuable for applications that must maintain lossless fidelity, while operating under stringent resource constraints, such as IoT nodes and wireless sensor

networks. The following subsections examine these emerging techniques, beginning with transform and residual based methods.

2.3.1 Transform and Residual Based Techniques

Transform and residual based approaches go far beyond the capabilities of dictionary and entropy coding. Since, they exploit correlations in the data before the final encoding stage. In transform coding, data are decorrelated so that maximum information is concentrated into a smaller number of coefficients, making following entropy coding like to be better in efficiency. Surveys by Sudhakar et al. (2005) and Rehman et al. (2014) evaluated ten wavelet transform based families, ranging from Haar to Biorthogonal, combined them with Set Partitioning in Hierarchical Trees (SPIHT) and Embedded Zerotree Wavelet (EZW). Their results displayed that applying SPIHT on Daubechies-4 could deliver a 10-15 dB gain in Peak Signal-to-Noise Ratio (PSNR) at 0.5 bits per pixel, corresponding to about a 3.5× reduction in image size which is amazing. In three-dimensional graphics, Peng et al. (2005) and Li et al. (2014) reported similar benefits by using codecs such as Edge Breaker and single-rate mesh schemes, which achieved transmission rates of less than two bits per vertex, which is sufficient for real-time web-based streaming of computer aided design models. Together, these studies demonstrated that how transform coding can efficiently handle the structured visual as well as the graphical data.

Residual coding provides a lighter alternative well suited for time-series data which is common in sensing and biomedical applications. Khelifati et al. (2019) introduced CORAD, a sparse dictionary-based scheme, it leveraged both temporal and cross-series correlations, achieving up to 40:1 lossless compression on IoT datasets, while it still guarantees online error bounds. Rasheed et al. (2021) proposed a log2 sub-band compression method for bio signals, he decomposed them into frequency sub-bands, and then encoded each band separately. This improved compression ratios and also retained its diagnostic fidelity, making it valuable in healthcare signal pipelines. Similarly, Mo et al. (2011) applied a hybrid of residual coding with adaptive Huffman coding to temperature signals, achieving efficient lossless compression on the sensor nodes with limited resources.

Other approaches have attempted to combine compression efficiency with robustness in noisy environments. Tan et al. (2010) designed a bit-error-aware scheme that used linear prediction followed by entropy or block-based residue coding, while protecting only key coding parameters with forward error correction. This balance allowed for high compression without compromising waveform integrity. Kiely et al. (2010) further advanced residual compression with Adaptive Linear Filtering Compression (ALFC), which used adaptive prediction and entropy coding of residuals. Implemented with integer-only arithmetic, ALFC allowed independent packet decoding and demonstrated higher compression ratios and better energy efficiency than S-LZW in seismic WSN deployments.

Dimensionality reduction techniques have also been explored as residual-inspired methods. Guo, Xie, and Jin (2018) presented a randomized SVD-based approach for compressing NMR data, showing that significant storage savings could be achieved without compromising accuracy. More recently, Hwang et al. (2023) proposed a hybrid multi-stage codec for waveform data that combined prediction and coding across multiple stages, providing flexibility for time-series compression in IoT and biomedical contexts.

Alongside these coding techniques, preprocessing transforms play a crucial role in boosting compression efficiency. Quantization methods, both scalar and vector, as well as predictive transforms, are often applied prior to coding. The Burrows–Wheeler Transform (BWT), introduced by Burrows and Wheeler (1994), reorders symbols in a sequence to enhance the efficiency of subsequent compression stages, particularly Run-Length Encoding (RLE) and Huffman coding. Combined compression pipelines such as BWT + RLE + Entropy coding, which are implemented in tools like bzip2, remain highly effective for structured text, scientific datasets, and sensor log compression. Beyond algorithmic pipelines, Sanborn and Ma (2005) argued that the autocorrelation function could be used to measure redundancy within datasets, guiding the choice of coding techniques and improving compressibility in structured or periodic signals.

While transform-based methods excel in multimedia applications such as images, video, and 3D graphics, they are less directly applicable to generic sensor data streams. Residual coding schemes, though more lightweight, are often not paired with semantic preprocessing or designed with hardware efficiency in mind. These limitations highlight a key motivation for this thesis: to combine residual coding with semantic preprocessing in order to achieve efficient entropy reduction under the strict resource and energy budgets of Wireless Sensor Networks (WSNs) and IoT platforms.

2.3.2 Domain-Specific Compression

Domain-specific compression methods have gained importance as applications increasingly demand solutions tailored to the properties of specific datasets. One of the most prominent areas is medical imaging, where compression must balance efficiency with strict requirements of diagnostic fidelity. Sridevi et al. (2012) reviewed schemes for medical image compression and emphasized that healthcare applications push codecs beyond generic performance metrics toward clinically safe compression. Butta et al. (2015) provided a complementary review of ECG data compression methods, again highlighting the critical role of preserving waveform features essential for diagnosis. Building on these reviews, Rebollo-Neira (2019) proposed a sparsity-based approach for ECG signals that achieved strong compression ratios with low distortion, demonstrating how signal structure can be exploited to enhance performance. The growing adoption of wearables and IoT-based

healthcare systems has further expanded research into lightweight biomedical codecs. For example, Chang and Sobelman (2024) developed a compact ECG compression scheme for medical IoT devices, while Sharma et al. (2022) implemented FPGA-based IQ-data compression to reduce latency and processing overhead at the edge. These studies illustrate how domain-specific medical codecs increasingly incorporate both algorithmic and hardware-level optimizations.

Beyond biomedical data, domain-specific methods have also been applied to semi-structured healthcare records. Pandey et al. (2020) presented a hybrid pipeline combining run-length encoding with infix and bit-reduction techniques, exploiting recurring clinical tokens in electronic health records to achieve notable storage savings. In a similar spirit of data-aware design, Subramanian et al. (2021) proposed a hybrid compression pipeline for 3D medical images that integrated Volume of Interest (VOI) extraction with LZW–Arithmetic coding, outperforming traditional Huffman and RLE while preserving diagnostic regions. Together, these works highlight how healthcare applications demand codecs that are not only efficient but also tuned to the unique statistical or semantic patterns of clinical data.

Domain-specific compression also extends to remote sensing and scientific imaging. Bartrina-Rapesta et al. (2017) designed a lightweight contextual arithmetic coder for satellite imaging, carefully balancing efficiency against the hardware limitations of on-board systems. Gundogar et al. (2018) introduced a hyperspectral image compression scheme based on tridiagonal matrix transformations, showing improvements in both storage and transmission for Earth observation datasets. Lin (2019) proposed early-stage lossless compression techniques for raw image acquisition, demonstrating benefits for initial image processing pipelines. These works show how compression for scientific data prioritizes reliability and hardware feasibility alongside efficiency.

Emerging application domains have also motivated customized codecs. Su et al. (2019) developed a DCT-based compression scheme for 5G vehicular networks, emphasizing real-time communication efficiency. Patel and Chaudhary (2017) investigated transform- and hybrid-based methods for wireless multimedia sensor networks (WMSNs), demonstrating how multimedia compression in resource-limited environments requires balancing fidelity with energy and memory constraints.

Text compression has likewise benefited from domain-aware approaches. Logeswaran (2002) demonstrated that combining a neural network predictor with arithmetic coding could reduce entropy beyond classical methods for English text. Jrai et al. (2015) improved Arabic text compression ratios by normalizing Unicode input before applying adaptive LZW, achieving gains from 58% to 71% on large corpora. These examples underscore how language-specific preprocessing can significantly improve compression efficiency in text domains. Expanding on language-specific optimizations, Amaeer et al. (2019) proposed an efficient

compression scheme for Bangla natural text by transforming 16-bit Unicode characters into an 8-bit representation. This Unicode reduction significantly minimized redundancy and achieved compression gains of 50–63%, outperforming standard Huffman and LZW algorithms. Their study highlights how exploiting linguistic and encoding characteristics can substantially improve compression ratios for regional language datasets.

Recent studies further illustrate the growing sophistication of script- and domain-aware pipelines. Suarjaya (2012) introduced *J-Bit Encoding*, a bit-level manipulation strategy that achieved 15–20% compression on hybrid multimedia files comprising audio, text, and video. Extending this concept to linguistic preprocessing, Amir et al. (2019) demonstrated that converting Bangla 16-bit Unicode text to an 8-bit representation yielded 50–63% compression gains, outperforming standard Huffman and LZW approaches. Ignatoski et al. (2020) compared arithmetic (entropy-based) and LZW (dictionary-based) coding across eight European languages, showing that LZW tends to outperform entropy-based methods in languages with higher redundancy and smaller alphabets.

Taken together, these studies reveal that domain-specific codecs achieve their effectiveness by leveraging data characteristics whether physiological signals, clinical records, satellite imagery, vehicular communication, or text corpora. While such approaches often remain narrowly tuned to their respective applications, they demonstrate the potential of combining preprocessing, classical methods, and hybrid pipelines to maximize redundancy removal in ways that general-purpose codecs cannot.

2.3.3 Hybrid and Adaptive Methods

The evolution of compression technology has steadily moved toward hybrid and adaptive frameworks designed to overcome the limitations of traditional single-algorithm codecs. In a hybrid design, two or more classical approaches are strategically combined so that the strengths of one method can compensate for the weaknesses of another. Boopathiraja and Kalavathi (2018) demonstrated this principle by integrating LZW and arithmetic coding into a unified compression pipeline for multispectral images, achieving markedly better efficiency than either technique on its own. Similarly, Tariq et al. introduced a serial hybridization strategy, where algorithms were applied in a specific sequence to maximise redundancy removal and improve overall performance. Their results showed that thoughtful ordering of algorithms can significantly influence compression effectiveness.

Beyond improving efficiency, hybrid frameworks have also been extended to enhance data security. Usama, Malluhi, and Zakaria (2021) developed a secure data-compression method that merges chaotic encryption with adaptive Huffman coding.

The scheme not only increased compression performance but also strengthened confidentiality, illustrating how hybrid designs can serve dual purposes in both data reduction and protection.

Hybridisation has further proven valuable for language-specific applications. Awajan and Abu Jrai (2015) proposed a hybrid model for Arabic text that combined multiple coding strategies to improve efficiency while preserving linguistic fidelity. Their work highlighted that hybrid codecs are adaptable across diverse domains, from imagery and sensor data to text with complex character sets. Along similar lines, Zheng et al. (2021) designed a hybrid pipeline uniting Huffman and LZW coding, achieving measurable gains in compression ratio on structured sensor datasets. However, their findings also pointed to a recurring challenge hybrid codecs tend to introduce added algorithmic complexity and require careful parameter tuning to maintain stability and speed.

Parallel to these developments, adaptive compression schemes have emerged to address data sources whose characteristics evolve continuously, such as those encountered in wireless sensor networks or real-time multimedia systems. Adaptive approaches dynamically modify their coding parameters in response to changes in data patterns, striking a balance between compression efficiency and computational cost. While adaptivity enhances responsiveness and optimises performance under varying conditions, it also introduces implementation challenges. High processing demands and limited on-chip resources often constrain their deployment in lightweight or energy-sensitive environments.

Taken together, hybrid and adaptive designs represent a natural evolution of compression research from fixed, single-stage models to intelligent, context-aware systems capable of adjusting to both data behaviour and system constraints.

2.3.4 Energy and Memory Constrained Environments

For embedded systems, Wireless Sensor Networks (WSNs), and Internet-of-Things (IoT) platforms, data compression must balance two critical constraints: reducing transmission volume while keeping on-node memory and energy consumption extremely low. Even simple, fixed-memory implementations have demonstrated significant benefits. For example, Sadler and Martonosi (2006) showed that Static-LZW on TinyOS motes, using a compact 256-entry dictionary, could deliver substantial energy savings compared to raw transmission. Marcelloni (2008) reinforced this principle by demonstrating that modest algorithmic simplifications and minimal bookkeeping could extend network lifetimes, underscoring how small design choices often lead to disproportionately large system gains.

Recent research has increasingly focused on adaptive compression techniques that trade modest local computational effort for significant reductions in network traffic. Knowledge-aware run-length encoding schemes, as proposed by Sharaff et al. (2022), effectively group semantically related runs to minimize payload sizes in structured and context-rich datasets. Similarly, predictor-based pipelines with periodically updated codebooks adapt to evolving signal statistics, achieving high efficiency on benchmark datasets such as the Intel Lab traces shows by Ketshabetswe et al. (2021). However, these adaptive schemes introduce a tension: model updates and probability estimation require extra CPU cycles and sometimes additional communications, which can offset compression-related energy savings in highly dynamic deployments (Lungisani et al., 2022). In this context, Ahmad et al. (2019) proposed a lossless compression algorithm designed for energy-efficient WSNs, demonstrating improvements in both compression ratio and energy consumption. Their work reinforces the importance of tailoring codecs to the tight energy budgets of embedded sensing systems.

Another line of work explores transform and aggregation techniques. Discrete Wavelet Transform (DWT) based codecs, for example, decorrelate frequency-rich environmental signals and achieve strong compression ratios, though their need for buffering and heavy arithmetic makes them impractical for low-end motes producing short, categorical payloads by Manuel et al. (2023). Mehta and Lobiyal (2017) highlighted that cluster-based data aggregation can effectively reduce communication overhead by exploiting spatial correlations among sensor nodes. However, approaches that rely primarily on coarse similarity measures often fail to capture the deeper semantic relationships inherent in sensor observations. In parallel, sparsity-driven techniques, such as distributed compressive sensing, have demonstrated strong potential by reducing transmission volume through the exploitation of joint sparsity across network nodes. Nevertheless, these methods typically shift the computational complexity to sink-side reconstruction and seldom provide interpretable, node-level grouping, as observed by Li et al. (2019).

In recent years, compression techniques originally developed for numerical and matrix-based workloads have been adapted for energy-aware data processing. For example, compressed sparse matrix-vector multiplication (SpMV) has demonstrated considerable potential for energy savings when implemented on hardware accelerators. However, as noted by Tosoni et al. (2025), existing evaluations have predominantly targeted numerical computation pipelines, with limited emphasis on the heterogeneous and multimodal sensor streams characteristic of modern IoT environments.

Across these family's dictionary and run-length methods, adaptive predictor plus entropy pipelines, transforms and clustering, and sparsity-driven approaches a common limitation emerges: there is little support for an inexpensive, general-purpose preprocessing stage that can inject domain semantics into the pipeline. Without such a stage, even efficient kernels must operate on raw, noisy

measurements, limiting their effectiveness in heterogeneous deployments. This gap suggests the need for lightweight, ontology-aware preprocessing layers that capture semantic structure at minimal on-node cost, thereby improving the performance of low-overhead codecs without adding significant computational or communication burdens.

2.3.5 Emerging Trends: AI-Integrated Compression

One of the most notable recent developments in data compression is the growing integration of machine learning and artificial intelligence (AI). Unlike handcrafted entropy coders, neural network models can learn probability distributions directly from data, often uncovering redundancies that traditional methods overlook. In some cases, these approaches have been shown to outperform classical algorithms. For example, Guorui Li et al. (2019) employed denoising autoencoders to enhance energy-efficient data collection in wireless sensor networks, demonstrating how learned representations can reduce transmission loads. In the healthcare domain, Hanoune and Lysmos (2020) investigated AI-inspired compression in an intelligent e-health gateway, showing that lightweight models could support medical monitoring without sacrificing robustness.

Extending this direction, Nasif et al. (2021) proposed a deep learning-assisted adaptive Huffman codec, where neural models guided codeword assignment. While the approach improved adaptability, its computational overheads limited deployment on severely resource-constrained devices. Building on this idea, Nasif et al. (2024) developed a deep learning–guided Huffman compression framework, reporting further gains in adaptability and compression ratio. However, as with many AI-driven methods, the increased processing requirements restricted applicability to lightweight IoT nodes.

More broadly, AI-based compression strategies are beginning to show promise in diverse areas, including video coding, anomaly detection in sensor data, and domain-specific biomedical applications. Yet, most remain specialized, relying on offline training or demanding significant memory and computational resources. These characteristics limit their feasibility in the constrained environments of WSNs and embedded IoT platforms.

This analysis highlights a clear research opportunity. While AI-driven codecs demonstrate strong adaptability and compression potential, what is missing is a generalizable, lightweight framework that can combine the efficiency of classical methods with the adaptability of machine learning, all within the practical constraints of real-world deployments. Addressing this challenge is one of the motivations underpinning the present thesis.

2.3.6 Concluding Remarks on Modern Codecs

Modern codecs build upon the foundations of classical and standardized methods but extend them through hybridization, adaptivity, and domain-specific customization. Unlike earlier approaches that primarily sought to maximize compression ratios, modern designs also account for the demands of specialized application domains such as healthcare, remote sensing, and low-power sensor networks. Their ability to tailor coding strategies to data characteristics and hardware environments marks a decisive shift from "one-size-fits-all" compression toward context-aware solutions.

At the same time, the growing integration of artificial intelligence and machine learning into compression pipelines signals a paradigm shift. Learning-based methods show strong potential to uncover redundancies beyond the reach of handcrafted algorithms and to adapt dynamically to changing data conditions. However, they remain constrained by training requirements, memory footprints, and computational overheads, which limit their deployment on lightweight embedded devices.

Taken together, these developments highlight both the progress and the remaining gaps. While modern codecs achieve impressive performance in specific contexts, they often lack generality across heterogeneous IoT deployments. This creates an opportunity for approaches that combine semantic preprocessing, residual coding, and lightweight adaptive mechanisms to deliver both efficiency and practicality. Addressing this need provides a central motivation for the research presented in this thesis.

2.4 Contemporary Codecs Evaluated in This Thesis

Alongside the classical algorithms and widely adopted standards, a number of modern codecs have been developed and optimized for practical deployment. Some of these are general-purpose, designed to perform reliably across diverse computing environments, while others are domain-specific, tailored for scientific, aerospace, or resource-constrained applications. The codecs selected for evaluation in this thesis represent this broad design spectrum and highlight key trade-offs between compression ratio, speed, memory usage, and suitability for embedded or low-power systems.

Comparative evaluations by Bhattacharjee (2013) provide a useful benchmark for situating these codecs. Their study systematically examined the performance of multiple lossless data compression methods and demonstrated how algorithmic families differ when applied to varying data characteristics. Such empirical baselines remain valuable when positioning newer codecs such as Zstandard (Zstd), Brotli, and

Adaptive Lossless Data Compression (ALDC), which are included in this thesis for detailed assessment.

Earlier comparative studies by Kodituwakku and Amarasinghe (2010) also shed light on the performance trade-offs among classical algorithms. Testing on text files of different sizes and types, they found that LZW delivered strong compression ratios for larger, more redundant datasets, while Run-Length Encoding (RLE) achieved high speed but offered poor compression for typical text. Huffman and Shannon–Fano methods were shown to offer balanced performance in terms of ratio and speed, whereas Adaptive Huffman introduced higher latency. A notable observation across these experiments was that compression time generally scaled with file size, except for RLE, which remained nearly constant. These results highlight how differences in algorithmic design translate into diverse operational profiles, providing context for evaluating contemporary codecs.

Recent innovations illustrate how contemporary methods push beyond classical trade-offs. Liu et al. (2018) designed a hardware-oriented modification of the LZ4 algorithm, enabling efficient deployment in device-level compression units, thereby bridging software efficiency with hardware feasibility. More recently, Barina (2022) introduced X3, a lossless data compressor that integrates modern entropy modeling with practical system considerations, establishing a new reference point for contemporary codec evaluation.

Collectively, these studies demonstrate the evolving balance between theoretical efficiency, practical performance, and system integration. By including Zstd, Brotli, ALDC, and related modern codecs in its evaluation, this thesis positions its analysis within the broader context of compression research—contrasting emerging methods with both classical baselines and standardized solutions.

2.4.1 General-Purpose Modern Codecs

Modern general-purpose codecs are designed to achieve strong compression while remaining practical across diverse applications such as cloud storage, web communication, and operating systems. They extend classical techniques like LZ77 with advanced entropy models, block transforms, and dictionary optimizations, striking different balances between compression efficiency, speed, and resource requirements. The following representative codecs are evaluated in this thesis as they capture the spectrum of design trade-offs in contemporary practice.

i. Zstandard (Zstd)

Introduced by Collet (2016), Zstd combines LZ77-style parsing with Finite State Entropy (FSE), a form of Asymmetric Numeral Systems for entropy coding. By

identifying repeated patterns through a sliding window and encoding symbols with probability-based codes, Zstd achieves excellent compression ratios while offering extremely fast decompression. Its adoption in large-scale storage systems and cloud platforms reflects its ability to balance efficiency with performance, making it one of the state-of-the-art codecs in current use.

ii. Brotli

Developed by Alakuijala and Szabadka (2016) for web and text compression, Brotli integrates LZ77 parsing, context modeling, and Huffman coding, supported by a static dictionary optimized for common web strings. It excels in compactly encoding HTML, CSS, and JavaScript resources, and its inclusion in HTTP standards has cemented its role as a widely deployed web codec. Brotli demonstrates how targeted dictionary design and entropy coding can make compression highly effective for specific application domains.

iii. Bzip2

As a transitional codec from earlier generations, bzip2 (Seward, 1996) illustrates the progression toward modern block-based methods. Its pipeline applies the Burrows–Wheeler Transform (BWT) to reorder symbols, followed by Move-to-Front (MTF) coding to cluster repeated symbols, and finally Huffman coding for entropy compression. While bzip2 achieves higher ratios than many early LZ-based methods, its relatively slow speed limits its adoption in time-sensitive applications. It is included in this evaluation primarily as a historical benchmark, showing how block-sorting techniques informed the design of later adaptive codecs.

iv. LZ4 and LZ4HC

Collet (2011) introduced LZ4 as an extremely lightweight implementation of the LZ77 family. It focuses on identifying short repeated substrings and replacing them with references, prioritizing speed over compression ratio. The high-compression variant, LZ4HC, performs deeper searches for longer matches, trading some performance for improved ratios. Because of their exceptionally fast throughput and low latency, LZ4 variants are widely deployed in databases, operating systems, and real-time systems. Their evaluation in this thesis emphasizes the role of speed in resource-constrained environments where rapid compression and decompression are often more critical than maximum savings.

v. LZMA (Lempel–Ziv–Markov Chain Algorithm)

Developed by Pavlov (1998), LZMA represents the opposite end of the spectrum. It extends the LZ77 framework by supporting very large dictionaries and employing range coding (a variant of arithmetic coding) to encode symbol probabilities with

high precision. This yields excellent compression ratios but at the cost of substantial memory use and longer processing times. LZMA is included here as a reference point for "maximum compression" approaches, providing a strong baseline against which more lightweight codecs can be assessed.

2.4.2 Domain-Specific Codecs

While general-purpose codecs are designed to operate across a wide variety of applications, many compression methods have been developed for specific domains, where system or data characteristics impose unique requirements. Such codecs are particularly relevant in scientific, aerospace, and resource-constrained environments, where compression must balance efficiency with strict hardware or energy limitations.

i. Rice Coding

Introduced by Rice (1979), Rice coding is a simplified variant of Golomb coding, optimized for geometric distributions such as prediction residuals. It encodes each number as a quotient-remainder pair, using unary representation for the quotient and binary for the remainder. Its computational simplicity and efficiency led to its adoption in astronomy (e.g., FITS format) and in CCSDS standards for space telemetry. Rice coding remains an important example of how lightweight entropy models can be tailored for hardware-constrained environments.

ii. FELACS (Fast and Efficient Lossless Adaptive Compression Scheme)

Developed within CCSDS recommendations (2005), FELACS was designed for satellite imagery and telemetry. It predicts pixel values and encodes the residuals using adaptive rules, thereby ensuring both speed and low memory consumption. FELACS is highly effective for onboard spacecraft compression, where hardware must process large imaging workloads under stringent constraints. Its inclusion here illustrates how adaptive prediction-based methods can deliver efficiency in domain-specific scientific missions.

iii. Adaptive Lossless Data Compression (ALDC)

ALDC extends dictionary-based compression with adaptive updates, allowing it to adjust to the statistical characteristics of the input stream. Initially applied in IBM storage systems, it has also been used in lightweight WSN deployments (Sadler & Martonosi, 2006). Kolo et al. (2012) advanced ALDC by partitioning sensor data into blocks and adaptively selecting code options for each block, achieving strong results on real-world WSN datasets. Later variants have emphasized memory optimization and energy efficiency, making ALDC a

particularly relevant codec for embedded sensing platforms. Its adaptability and low resource requirements align closely with the objectives of this thesis.

Taken together, domain-specific codecs such as Rice, FELACS, and ALDC demonstrate how compression schemes evolve when designed around the constraints of specialized environments whether spacecraft imaging systems, scientific data pipelines, or sensor nodes. However, despite their strengths, none of these codecs simultaneously achieve high compression efficiency, minimal memory use, and energy awareness across diverse IoT deployments. This limitation underscores the need for new approaches that can bridge these trade-offs, providing the motivation for the work presented in this thesis.

Contemporary codecs such as Zstd, Brotli, LZMA, Rice, FELACS, and ALDC represent the state of the art in modern compression. However, none of these approaches simultaneously deliver high compression efficiency, ultra-low memory footprint, and energy awareness requirements that are critical for Wireless Sensor Networks and IoT platforms. This limitation highlights the need for new approaches that integrate residual and semantic preprocessing with lightweight, hardware-friendly coding strategies. Addressing this challenge forms a central focus of the work undertaken in this thesis.

2.5 Hardware and System Level Implementations

While software codecs offer flexibility and ease of deployment, implementing compression at the hardware or system level enables real-time throughput and energy-efficient operation capabilities that are especially critical in embedded and IoT contexts. Hardware acceleration reduces latency, minimizes software overhead, and allows codecs to function effectively under the stringent performance and energy constraints of edge devices.

Early work in this area focused on FPGA and ASIC accelerators for classical algorithms. Rigler et al. (2007) demonstrated one of the first FPGA-based implementations of a combined Huffman and LZ77 codec for GZIP compression, showing how general-purpose lossless codecs could be re-engineered to achieve hardware-level speedups. Such efforts established the feasibility of embedding compression logic directly into hardware to meet real-time requirements.

Subsequent research broadened the scope to include memory- and cache-resident compression mechanisms. Mittal and Vetter (2016) examined GPU- and cache-level compression, reporting reductions of 15-20% in off-chip memory traffic with minimal latency overhead. These results underscored the potential of

compression not only as a data-storage or transmission tool but also as a system-level optimization technique for reducing bandwidth bottlenecks.

More recent studies have analysed hardware compression from a system-architectural perspective. Carvalho and Seznec (2021), for example, surveyed cache compression techniques and evaluated not only compression ratio and throughput but also associated trade-offs such as tag overhead, block placement, and decompressor latency. Their work highlights the complexity of integrating compression into modern memory hierarchies: while throughput gains are often significant, inefficiencies in cache design or decompression energy can offset the benefits.

Collectively, these studies illustrate that successful hardware-level deployment of compression requires both microarchitectural detail and system-level coordination. FPGA/ASIC accelerators demonstrate the raw speed potential of classical methods, while cache and memory compression research shows how integration into broader architectures can extend benefits to system performance and energy efficiency. Taken together, this body of work confirms that hardware-aware design is a crucial component of real-world compression solutions, particularly for embedded, IoT, and high-performance platforms.

2.5.1 FPGA and ASIC-Based Implementations

Hardware accelerators for compression have been extensively investigated to achieve real-time throughput and energy efficiency, particularly in embedded systems where software-based methods may be too slow or power-hungry. Early contributions established the foundation for hardware algorithm co-design, demonstrating how codec design could be tightly integrated with circuit-level optimization. For example, Min-Bo, Lee, and Jan (2006) presented a lossless compression algorithm together with a dedicated hardware architecture, highlighting how compression efficiency could be enhanced by considering hardware constraints during algorithm development.

Building on these principles, subsequent research focused on FPGA-based implementations that emphasized either optimization or security. Jeevan et al (2014) implemented a secure image compression system in Verilog HDL, integrating compression with on-chip data protection. Around the same time, Dhawale (2014) optimized Huffman coding in VHDL, demonstrating that low-level architectural improvements could significantly boost throughput on FPGA platforms. Teja et al. (2015) extended these efforts by designing a VLSI implementation of a multiplier-less 2D DCT/IDCT, enabling efficient transform coding for real-time image compression.

More recent advances have aimed at maximizing throughput and aligning with international standards. Guguloth and Vadtya (2025) proposed a high-throughput canonical Huffman codec with parallel frequency counting and pipelined codebook generation, achieving state-of-the-art real-time performance on Xilinx FPGAs. Similarly, Barrios et al. (2021) realized the CCSDS 123.0-B-2 standard via high-level synthesis (HLS), attaining near-lossless compression at 12.5 MS/s with minimal hardware resources, the first fully compliant FPGA realization of this standard. Hameed et al. (2018) contributed a low-power Huffman text compressor on Altera platforms, reporting significant energy savings. At the VLSI level, Sadaghiani and Ghanbari (2019) proposed an optimized compression processor based on a modified Loeffler architecture, achieving high-speed operation with reduced complexity.

Domain-specific applications have also benefited from hardware implementations. For instance, Kamran Rasheed et al. (2021) developed a Verilog-based lossless compression system for medical signals, showing that FPGA-based pipelines can meet the stringent latency and energy requirements of biomedical monitoring systems.

Overall, FPGA and ASIC implementations have demonstrated that throughput and efficiency can be significantly enhanced when algorithms are adapted for hardware. However, these designs are typically optimized for fixed algorithms or specific standards, limiting their adaptability to the heterogeneous and dynamic conditions of IoT deployments. This underscores the need for lightweight, reconfigurable kernels that can combine hardware-level efficiency with algorithmic flexibility.

2.5.2 Memory-System Compression

Beyond FPGA and ASIC accelerators, compression has also been integrated directly into memory subsystems as a strategy to alleviate bandwidth bottlenecks and reduce off-chip traffic. Unlike software or peripheral hardware implementations, memory-system compression treats compression as a built-in feature of the memory hierarchy, thereby enhancing both bandwidth and effective capacity without requiring major redesign of the processor core.

Early survey efforts provided a broad understanding of these architectural possibilities, emphasizing how compression could serve as a transparent layer in cache and memory subsystems. Their analyses highlighted that carefully designed schemes have the potential to improve effective memory bandwidth and capacity while incurring only modest hardware modifications.

Building on this foundation, as previously discussed, Mittal and Vetter (2016) also provided a detailed survey of cache-resident compression techniques, detailing practical implementations across high-performance computing environments. Their results demonstrated that memory-system compression can reduce off-chip traffic by 15–20% while adding less than 5% latency overhead, underscoring its suitability for bandwidth-constrained, performance-critical systems. More specialized implementations have also been explored. For example, Mahdavi et al. (2019) proposed an area-efficient on-chip compression scheme for massive MIMO systems, showing that compressing channel data can significantly reduce storage overhead in high-throughput communication pipelines.

Together, these studies confirm that memory-system compression provides tangible benefits in terms of bandwidth, storage efficiency, and performance. However, such designs are generally optimized for large-scale computing platforms and assume relatively abundant resources. Their reliance on sophisticated memory controllers and ample processing power makes them less directly applicable to ultra-constrained platforms such as WSNs and IoT nodes, where strict memory and energy budgets dominate design considerations. This gap highlights the need for lightweight, memory-aware compression schemes that can deliver efficiency comparable to system-level approaches, but within the stringent constraints of embedded environments.

2.5.3 Edge and Sensor-Level Integration

At the sensing edge, compression can be co-designed directly with data acquisition systems to minimize redundancy at the earliest stage of signal capture. Trakimas and Sonkusale (2011) introduced an adaptive-resolution asynchronous analog-to-digital compression (ADATA) architecture for energy-constrained sensing platforms. Fabricated in 0.18 µm CMOS, their design dynamically adjusted sampling resolution according to signal activity, thereby achieving compression during data acquisition itself. The prototype reached a maximum sampling rate of approximately 50 KS/s with a signal-to-noise-and-distortion ratio (SNDR) of 43 dB while consuming only 25 µW. This work demonstrated that joint ADC compression architectures can enable ultra-low-power operation in sensor nodes.

Beyond data acquisition, compression has also been integrated with security and communication frameworks. Bartrina et al. (2017) presented an IoT-Edge-Cloud pipeline that combined compression with AES-128/HMAC-based security. Their implementation achieved transmission energy savings between 32% and 90% while maintaining image quality above 92 dB PSNR, showing how compression can complement encryption and authentication to support secure, energy-aware data transfer in IoT deployments. Together, these studies highlight how sensor- and edge-level co-design enables compression to be embedded seamlessly into acquisition and

communication pipelines, thereby optimizing energy usage and transmission overhead.

While hardware and system-level implementations demonstrate impressive throughput and energy efficiency, they largely focus on accelerating existing codecs or standards without integrating semantic or machine-learning–based adaptability. This limitation points to the need for lightweight, FPGA-friendly kernels enhanced with semantic awareness, capable of adapting to heterogeneous sensor environments. Addressing this challenge aligns with the objectives of this thesis.

2.6 WSN and IoT-Specific Compression

Wireless Sensor Networks (WSNs) and Internet of Things (IoT) platforms impose unique constraints on data compression owing to their limited memory, processing power, and strict energy budgets. To address these challenges, research has produced a wide spectrum of lightweight and adaptive schemes that trade complexity for efficiency while exploiting the structural properties of sensor data.

2.6.1 Distributed and Local Compression

Srisooksai et al. (2012) analyzed both distributed and local compression approaches, showing that collaborative in-network strategies can significantly reduce transmission energy by leveraging spatial and temporal correlations among nodes. Singh (2019) extended this direction by studying Distributed Compressive Sensing (DCS), where sparsity-aware sampling across nodes reduced data volumes substantially. While effective in minimizing transmissions, standard DCS formulations required complex reconstruction at the sink and did not provide compact on-node storage for repetitive arrays, limiting their practicality in constrained deployments. Liang and Li (2014) contributed a robust WSN compression algorithm that emphasized resilience against noise and packet loss, ensuring that lightweight nodes could achieve reliable performance under adverse conditions.

2.6.2 Wireless Multimedia Sensor Networks (WMSNs)

Image and multimedia compression has received dedicated attention in resource-limited WSN settings. ZainEldin et al. (2015) surveyed image compression techniques for WMSNs, focusing on how fidelity can be balanced against constrained resources. Rekha and Samundiswary (2015) introduced adaptive image compression methods that reduced power consumption using predictive schemes, while still maintaining acceptable visual quality. These contributions highlight the unique trade-offs in multimedia IoT deployments, where compression must preserve data fidelity while reducing both transmission cost and processing overhead.

2.6.3 Lightweight Adaptations of Classical Algorithms

Several studies have revisited classical algorithms for WSN contexts by tailoring them to lightweight environments. Gupta et al. (2022) adapted Huffman coding for WSNs, demonstrating that frequency modeling at the node or cluster level improves efficiency. However, they also noted that frequent model updates can offset energy gains in dynamic environments. Roy and Chandra (2020) evaluated LZW on sensor traces, showing moderate compression for repetitive payloads, but limited overall energy savings due to dictionary maintenance overhead. Pushpalatha and Shivaprakasha (2020) tested RLE variants, confirming their computational simplicity and suitability for bursty signals, but pointing out the lack of stateful history reduced overall efficiency. Together, these works underline how adapting classical methods requires careful balance between computational simplicity and network-level benefits.

2.6.4 Biomedical and IoT-Specific Applications

WSN and IoT compression research has also been shaped by domain-specific demands, particularly in healthcare. Hanoune and Lysmos (2020) proposed compression mechanisms within an intelligent e-health gateway, integrating data reduction with secure monitoring applications. Yan et al. (2021) designed a QRS detector for biomedical IoT systems that combined compression with time-sharing architectures, offering dual benefits of signal analysis and energy-efficient transmission. More recently, Qiu et al. (2025) developed a robust topology-generation and data-transfer scheme to sustain reliability in IoT deployments. While not explicitly focused on compression efficiency, the approach complements existing work by addressing data delivery integrity.

2.6.5 Domain-Specific Novel Approaches

Innovations have also emerged in body sensor networks and metadata management. Wu and Yu-Chee (2011) demonstrated that exploiting both temporal and spatial correlations in body area networks (e.g., Pilates motion recognition) could reduce redundancy while preserving recognition accuracy. Thomas and Mathew (2016) applied a quadtree-based clustering method to compress sensor node address metadata, reducing communication overhead without compromising data integrity.

Taken together, these studies demonstrate the diverse strategies applied to compression in WSN and IoT contexts from distributed sensing and multimedia pipelines to lightweight adaptations of classical methods and biomedical applications. Despite notable progress, existing methods often remain specialized, narrowly tuned, or limited by reconstruction costs at the sink. This highlights the need for generalizable, energy-aware frameworks that integrate lightweight coding with

semantic preprocessing, enabling efficient deployment across heterogeneous IoT platforms.

2.6.6 Baseline IoT Approaches

Alongside recent IoT-specific compression techniques, a number of earlier approaches continue to serve as important baselines in the literature. These methods illustrate how adaptive principles were first introduced into compression frameworks for constrained environments.

One such approach is the Adaptive Data Compression Algorithm (ADCA), which varies the compression degree in real time according to available bandwidth and processing capacity (Mohamed et al., 2018). This flexibility enables nodes to respond to changing network and resource conditions, making it useful in environments where traffic and load fluctuate. Similarly, Real-Time Compression Quality Adjustment (RCQA) adapts error bounds dynamically based on the sensitivity of the data stream (Shoaib et al., 2015). By preserving high fidelity for critical information while allowing less vital streams to be compressed more aggressively, RCQA introduced the idea of content-aware trade-offs between fidelity and efficiency. A further reference point is the Stream Control Transmission Protocol (SCTP), which integrates compression with encryption and signing within a transport layer framework (Lu et al., 2016). Although originally intended as a reliable transmission protocol, its incorporation of compression and integrity mechanisms demonstrated how system-level protocols could combine multiple objectives—data reduction, security, and reliability within a single pipeline.

These baseline methods are frequently used as benchmarks for evaluating modern IoT frameworks. While they provided valuable stepping stones and influenced subsequent designs, their performance in compression fidelity and energy efficiency remains modest when compared with contemporary schemes. Moreover, they often assumed processing and memory resources greater than those available in typical sensor nodes, limiting their portability to ultra-constrained WSN deployments.

Although effective in concept, most WSN and IoT compressors including baseline approaches such as ADCA, RCQA, and SCTP tend to exploit either temporal or spatial redundancy in isolation and rely on resource budgets larger than the sub-16 KB memory typical of embedded nodes. This restricts their real-world applicability, highlighting the need for ultra-lightweight solutions that jointly exploit spatio-temporal redundancy while remaining compatible with the strict memory and energy constraints of WSNs and IoT platforms.

2.6.7 Concluding Remarks WSN and IoT-Specific Compression

Research in WSN and IoT compression has primarily focused on minimizing communication overhead and energy consumption through lightweight coding and data aggregation. Classical methods such as Huffman, LZW, and RLE have been adapted for constrained sensor nodes, while predictive and transform-based approaches have improved compression ratios for specific data types. These developments highlight the continuing effort to achieve reliable compression without compromising the limited computational and memory resources typical of embedded systems.

Despite these advances, most existing techniques remain domain-specific and rely on complex reconstruction or limited correlation modeling. There remains a clear need for a generalized, energy-efficient, and memory-optimized compression framework that can exploit spatio-temporal and semantic redundancies across heterogeneous sensor data. Addressing this gap provides the foundation and motivation for the research framework developed in the subsequent chapters.

2.7 Machine-Learning and Autoencoder Methods

The growing availability of computational resources and sensor data has led to increasing exploration of machine learning (ML) and data-driven methods for compression. Unlike classical approaches, which rely on predefined statistical models, ML-based methods can learn patterns and redundancies directly from data, enabling more adaptive compression across heterogeneous sources.

Jayasankar et al. (2021) provided a broad survey categorizing compression methods by coding scheme, data type, and application domain. Their analysis highlighted a clear trend: recent approaches increasingly incorporate machine learning to balance compression ratio, data fidelity, and computational cost, particularly in complex IoT environments.

Specific techniques have demonstrated the potential of ML-driven compression in diverse domains. Guo et al. (2019) proposed a fast randomized Singular Value Decomposition (SVD) framework for nuclear magnetic resonance (NMR) datasets, applying Hadamard-based sketching before SVD to accelerate compression and inversion while maintaining accuracy. Zhao et al. (2018) developed an on-chip neural compression framework based on compressed sensing with sparse matrices, showing that ML-inspired methods can be mapped into hardware for practical deployment. Malhotra and Singh (2021) advanced this integration further by implementing decision tree classifiers on FPGA to selectively trigger compression, demonstrating how lightweight ML models can support hybrid hardware algorithm pipelines for energy savings.

While machine-learning based methods show considerable promise achieving high compression ratios and reducing transmission costs they often depend on offline training and lack guarantees on worst-case latency or resource use. This makes them less suitable for ultra-constrained, real-time deployments typical of WSNs and IoT platforms. These limitations highlight the need for lightweight, real-time semantic preprocessing layers that can adaptively guide compression without the computational burden of large ML models. Addressing this gap forms one of the central objectives of this thesis.

2.8 Application-Specific and Emerging Solutions

Beyond general-purpose codecs, a range of emerging solutions have been tailored to specialized domains and application specific requirements. These works illustrate how compression can be optimized for particular workloads, hardware environments, or communication pipelines.

In the area of high-throughput hardware design, Bruni and Johansson (2020) developed DPTC, a configuration-free FPGA-based trace compressor operating at 400 MS/s. Their method applied difference prediction and packed groups of four samples into 32-bit words, achieving 4-5 bits per sample within a low-latency, VHDL-optimized architecture. This demonstrated the feasibility of achieving compression at extreme sampling rates with minimal overhead.

For wireless sensor networks, Mishra et al. (2022) compared RLE, Adaptive Huffman, and LZ4 across real-world datasets, analyzing trade-offs between compression ratio and energy consumption. Their findings showed that while LZ4 offered stronger ratios, RLE remained the most energy efficient, positioning it as a practical choice for edge-sensor pipelines where energy budgets dominate.

Compression has also been combined with security and hybrid optimization frameworks. Liu et al. (2021) proposed a multi-stage pipeline that integrated LZW and Huffman compression with Physical Unclonable Function (PUF)-based key generation. Their hardware-assisted design achieved 473 Mbps throughput and a compression ratio of 0.558, while simultaneously providing resilience against cloning attacks an example of compression coupled with cryptographic robustness. Similarly, Oswald and Sivaselvan (2018) explored hybrid pattern-mining and clustering pipelines combined with RLE and Huffman coding, reporting compression improvements between 18% and 751% relative to single-method baselines.

In the context of next-generation communication systems, Lee et al. (2019) proposed an FPGA-compatible IQ compression scheme for LTE and 5G fronthaul links. Their design used cyclic prefix/suffix adjustments, decimation, bit-width truncation, and partial Huffman coding to achieve nearly 4× compression with only

an approximately 2% increase in Error Vector Magnitude (EVM). This demonstrated that compression could be tightly integrated into fronthaul pipelines without significantly affecting signal integrity.

Taken together, these studies illustrate how application-specific designs can deliver impressive domain-level gains, whether in terms of energy efficiency, hardware throughput, or integration with security and communication standards. However, they are often monolithic and tightly coupled to particular datasets or platforms, limiting generalizability. This underscores the need for a modular, ontology-aware compression framework that can flexibly adapt across heterogeneous environments a central direction pursued in this thesis.

2.9 Overall Synthesis

Overall, the surveyed literature demonstrates the evolution of compression research from its theoretical foundations and classical codecs to standardized formats, hybrid pipelines, domain-specific methods, and, more recently, AI-driven approaches. These studies highlight significant progress in enhancing compression efficiency, computational performance, and application-oriented customization.

At the same time, they also expose persistent limitations: inadequate support for stringent memory and energy constraints in WSNs, trade-offs between throughput and memory usage in hardware-oriented platforms, and the absence of lightweight semantic integration in many modern frameworks. Taken together, these recurring challenges underscore the unresolved issues in the field. The next chapter consolidates these observations into well-defined research gaps, which form the foundation for articulating the objectives of this thesis.

2.10 Research Gaps

Despite extensive progress in the field of data compression, significant limitations remain when these techniques are applied in real-world, resource-constrained environments. The existing body of work demonstrates strong advances in compression ratio, algorithmic efficiency, and domain-specific optimization, yet recurring challenges such as high memory demands, limited adaptability, fragmented hybrid designs, and inadequate consideration of energy and latency persist. These shortcomings hinder the seamless deployment of compression in IoT, wireless sensor networks (WSNs), and embedded platforms. To address these challenges systematically, the following research gaps are identified, each of which directly motivates the objectives of this thesis.

2.10.1 Context-Agnostic Encoding

Most compressors from classical Huffman and LZW to modern Zstd, Brotli, and LZ4 apply uniform strategies regardless of whether the input data is smooth, bursty, or periodic. This “one-size-fits-all” approach limits efficiency, as different signal patterns demand specialized coding strategies.

2.10.2 Memory–Throughput Trade-Off

High-performance hardware codecs deliver hundreds of Mbps throughput but require tens of kilobytes of RAM, which exceeds the sub-16 KB memory budgets of many IoT motes. On the other hand, ultra-lightweight schemes fit within kilobyte footprints but offer only modest compression and lack adaptability to diverse data.

2.10.3 Fragmented Hybrid Solutions

Hybrid pipelines (e.g., BWT→RLE→Huffman, or Zstd→LZ4HC) can significantly boost ratios. However, existing studies configure these manually for specific datasets. No unified framework exists for automatically adapting multi-stage pipelines across heterogeneous data types.

2.10.4 Ad-Hoc Semantic Preprocessing

Domain-aware preprocessing such as Arabic text normalization or ECG feature extraction has shown substantial benefits, but remains task-specific. A generalizable, ontology driven preprocessing framework is lacking to guide compressors toward the most informative segments of data.

2.10.5 Narrow Evaluation Metrics

Most research emphasizes compression ratio while neglecting latency, energy consumption, and memory footprint. For real time IoT deployments, compression gains are irrelevant if they come at the cost of prohibitive delays or excessive power draw.

2.10.6 Siloed Benchmarks

Benchmark datasets are often siloed by modality focusing on text, images, or sensor traces in isolation. Without unified testbeds that combine smooth signals, noisy bursts, periodic streams, and multilingual data, it is impossible to compare compressors fairly across use cases.

2.10.7 Limited On-Device Intelligence

Recent ML-based compressors and autoencoder schemes achieve impressive results offline, but depend on heavy training and lack worst-case performance guarantees on constrained hardware. Few approaches exploit lightweight, real-time classifiers or anomaly detectors to dynamically select the most efficient codec for current data.

Together, these gaps reveal that existing methods are either too generic, too resource-hungry, or narrowly optimized. What is missing is a unified, context-aware framework that balances compression efficiency with memory, speed, and adaptability in resource constrained environments. This thesis addresses that need by combining semantic preprocessing, residual and hybrid coding, and AI-assisted hardware implementations into a coherent compression architecture.

In this chapter, a comprehensive review of data compression research has been presented, spanning from the classical foundations of Shannon's entropy and early codecs to contemporary advances in standard formats, hybrid methods, and AI-integrated approaches. Special attention has been given to compression in constrained environments such as wireless sensor networks and IoT platforms, where memory, energy, and real-time operation pose unique challenges. Across these works, it is evident that each generation of techniques prioritized different objectives compression efficiency, speed, adaptability, or hardware feasibility yet no single approach fully addresses all the requirements of modern embedded and data-intensive systems. On the basis of these identified gaps, the objectives of the thesis were formulated, which in turn shaped the methodology. The next chapter presents the methodological framework developed to address these gaps.

CHAPTER 3

METHODOLOGY

This chapter builds on the research gaps discussed earlier and sets out the framework developed to tackle the shortcomings of existing compression methods. The approach combines semantic preprocessing, residual and hybrid coding strategies, and lightweight AI-assisted modules, all designed with the strict memory, energy, and processing limits of Wireless Sensor Networks (WSNs) and IoT devices in mind. The discussion begins with the overall design philosophy and workflow, and then moves on to describe the algorithmic framework, the datasets used, the evaluation metrics, and the implementation details. Each part of the methodology is closely linked to the challenges identified in the review, ensuring that the solutions are firmly grounded in the problems they are meant to address.

3.1 Research Design

The research was organised into six sequential phases, each addressing a specific objective and building upon the outcomes of the previous stage. This structured progression ensured a logical flow from baseline benchmarking to the development of domain-aware, hybrid, and hardware-assisted compression frameworks. The overall design is exploratory, developmental, and experimental: the exploratory stage involved benchmarking existing algorithms on diverse datasets, the developmental stage focused on designing novel compression frameworks, and the experimental stage validated these approaches through simulations and FPGA-based implementations. Together, these phases ensured that the contributions of this thesis are both scientifically rigorous and practically deployable. The sequential flow of the six phases is illustrated in Figure 3.1, which provides a visual overview of how the research progressed from baseline benchmarking to advanced hardware-assisted frameworks**.**

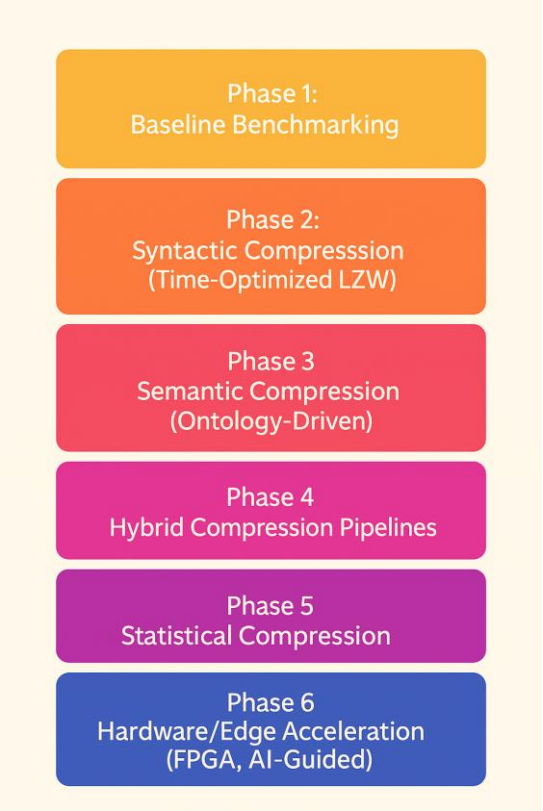

Fig.3.1: Sequential Flow of the Research Methodology across Six Phases

3.2 Methodological Phases

The research advanced in six connected phases, where the outcome of each stage shaped the direction of the next. This design ensured a smooth flow from benchmarking existing techniques to the development of syntactic refinements, semantic-driven methods, hybrid pipelines, residual-based frameworks, and finally hardware-assisted solutions. A concise overview of these phases is provided below, with detailed methods and results discussed in Chapters 4 to 9.

Phase I: Baseline Benchmarking

The study began with a comprehensive baseline analysis of standard compression algorithms such as RLE, Huffman, LZW, Zstd, LZ4, and LZMA. These were evaluated on diverse datasets, including sensor logs, synthetic traces, and multilingual text. This phase established reference points for compression ratio, throughput, and latency, while also highlighting the limitations of off-the-shelf methods in IoT and WSN environments.

Phase II: Syntactic Compression (Time-Efficient Dictionary-Based)
To address these gaps, the second phase introduced a syntactic refinement in the form of LZWP, a dictionary-based scheme optimised for throughput. By modifying dictionary update policies, LZWP achieved faster operation without major sacrifices in compression ratio. This phase demonstrated that classical compressors could be reshaped for resource-constrained platforms by improving efficiency at the syntactic level.

Phase III: Semantic Compression (Ontology-Driven)
Building on this, the third phase shifted focus from syntax to semantics. Healthcare data was normalised into RDF triples, while WSN readings were grouped with lightweight ontologies. A specialised method, OntoRLE, combined ontology-guided clustering with run-length encoding. This approach showed that embedding semantic knowledge into the pipeline could significantly improve compressibility while retaining contextual meaning.

Phase IV: Hybrid Compression Pipelines
The fourth phase combined insights from syntactic and semantic methods by designing multi-stage pipelines such as Zstd→LZ4HC. These were applied to Indic-language corpora and IoT text streams, demonstrating that carefully composed cascades could provide a better trade-off between compression ratio, speed, and decompression efficiency than single techniques.

Phase V: Residual–Canonical Framework (MOR-ALDC)
With the hybrid approach established, the fifth phase turned to WSN-specific optimisation. MOR-ALDC combined residual transformation, which captures temporal redundancy, with canonical Huffman coding to minimise memory and computation overhead. This framework was tailored to sensor networks where energy and memory are scarce, complementing the general-purpose hybrids of Phase IV.

Phase VI: AI-Guided FPGA and Edge Framework
The final phase focused on deployment. A lightweight AI decision module was developed to determine when compression should be applied, balancing computation cost with bandwidth and energy savings. MOR-ALDC was implemented in VHDL and tested on FPGA hardware, confirming that adaptive, energy-aware compression could operate in real time at the edge.

Together, these six phases form a connected research flow: starting with benchmarking, moving into syntactic refinements, semantic enrichment, and hybrid pipelines, then advancing to residual-based optimisation, and finally proving feasibility through AI-guided FPGA deployment.

3.3 Experimental Setup

The experimental setup was planned to include a wide range of codecs so that the evaluation would be both fair and comprehensive. Standard and well-known algorithms were first considered to establish reliable benchmarks. In addition, the study incorporated several advanced and customized approaches, such as LZWP, MOR-ALDC, AIoT MOR-ALDC, OntoRLE, and hybrid pipelines, each representing different design philosophies. Bringing these methods together allowed for a balanced comparison between conventional techniques and the proposed frameworks, highlighting both their strengths and limitations. All experiments were carried out under consistent protocols for dataset preparation, timing, memory profiling, and performance assessment to ensure reproducibility and meaningful results.

3.3.1 Tools & Environments

A Python-based benchmark harness was developed to systematically evaluate each codec. The framework automatically supplied input data to the compressors, measured encoding and decoding times, monitored memory usage, and recorded performance metrics such as compression ratio, throughput, and resource utilization for subsequent analysis.

Algorithm prototyping for semantic compression was carried out in MATLAB, while ontology-based preprocessing frameworks were designed using Protégé. Packet-level IoT traffic traces were collected and analysed using standard packet capture tools (e.g., Wireshark). All software-based experiments were executed on standard personal computing environments available during the course of the research. For hardware validation, compression kernels based on residual transformation and canonical Huffman coding were implemented in VHDL and synthesized using Xilinx Vivado/Vitis on a Zynq-7020 FPGA board. This setup enabled a complete software–hardware evaluation of the proposed methods under realistic constraints.

3.3.2 Datasets

A diverse set of datasets was used to ensure that the proposed compression techniques were evaluated across both controlled synthetic cases and real-world applications. Each dataset was chosen not only for its availability but also for its

ability to stress-test specific aspects of the proposed framework. The datasets are described below:

(i) English text corpora:

Standard benchmark collections, including the Canterbury Corpus and the Calgary Corpus, were employed. These corpora contain representative English text, source code, and document files of varying sizes and are widely used in compression research. They were selected to provide a standard baseline for evaluating compression ratio and speed against widely recognized reference material.

(ii) Hindi Text Corpora:

Devanagari-encoded Hindi stories were compiled from publicly available online sources. To evaluate scalability, the corpus was prepared in three representative sizes-145 KB, 1.6 MB, and 13 MB capturing small, medium, and large workloads. These datasets were designed to assess how well the compression schemes handle redundancy patterns distinct from English text, thereby testing their robustness across languages.

(iii) Hindi Bible Dataset:

The publicly available Hindi Bible Dataset (Verma, 2021) was obtained from Kaggle. This dataset provides continuous text in Hindi and offers a standardized benchmark for evaluating compression performance on structured multilingual corpora.

(iv) Healthcare streams:

Twenty patient-monitoring logs from Kaggle (Sudarshan, accessed 2023) were used for ontology-driven compression experiments. The dataset includes patient-related information such as visits, health camps attended, daily activities (e.g., blood pressure, heart rate monitoring), and treatment records. Its hierarchical structure makes it suitable for ontology-based modeling and semantic preprocessing. This dataset was selected to validate the ability of the framework to handle temporal healthcare data, a domain where data integrity and semantics are especially critical.

(v) Synthetic sensor signals:

Artificial datasets were generated to model smooth, bursty, and noisy signal patterns. These were used to deliberately stress-test the classification and adaptability of the framework under controlled conditions, ensuring that the algorithms can handle diverse real-world signal dynamics.

(vi) Wireless sensor traces:

Real-world WSN data was obtained from the Intel Berkeley Research Lab (Intel Lab, accessed 2025) and the UCI Air Quality repository (De Vito et al., accessed 2025). These datasets represent common environmental monitoring workloads. They were selected to validate the practical feasibility of the proposed methods in realistic sensor network deployments, where energy and memory constraints are strict.

(vii) Multilingual concept graph:

The AGROVOC Thesaurus (Food and Agriculture Organization, 2023) was incorporated as a structured multilingual ontology to guide semantic preprocessing and concept-based clustering of sensor and textual data. AGROVOC includes over 41,000 concepts with semantic linkages. This resource was chosen to demonstrate ontology-driven compression, allowing evaluation of semantic clustering methods for structured multilingual and cross-domain datasets.

(viii) Edge (packet-level) dataset:

For the edge-computing experiments (Paper 4), a packet-level dataset was generated experimentally rather than drawn from an existing repository. Network traffic was captured using Wireshark to record realistic IoT transmissions across a small testbed of devices; a representative Wireshark screenshot verifying the capture procedure is included as Figure 5.1. The collected packets were grouped into four categories to cover a range of IoT applications:

a) *Image data*: representing surveillance and healthcare imaging, where fidelity is crucial.
b) *Sensor data:* representing real-time machine-to-machine telemetry typical of IoT.
c) *Financial data:* representing high-integrity transactional streams where strict losslessness is essential.
d) *Textual data:* representing logs and communication streams, often semi-structured or unstructured.

This dataset was selected to mirror the heterogeneity of IoT workloads and test the framework across multimedia, environmental, financial, and communication domains.

Together, these datasets provide a comprehensive platform for evaluation. By combining standard text corpora, multilingual datasets, synthetic and real-world sensor traces, healthcare records, structured ontologies, and experimentally generated IoT packet data, the framework was tested under diverse conditions ranging from controlled benchmarks to heterogeneous real-world workloads. This diversity

ensures that the proposed methods are assessed not only for compression ratio, but also for scalability, adaptability, and feasibility in practical deployment scenarios.

3.3.3 Timing protocol

Compression timings were recorded using the Python implementation with a high-resolution wall-clock timer (time.perf_counter()). Each reported value is the arithmetic mean of five independent runs and, unless stated otherwise, includes the preprocessing step and the encoding pass. Timings exclude network transfer and any delays beyond local file input/output (I/O). The corresponding system and timing details are summarized in Table 3.1.

3.4 Software and Hardware Environment

To ensure reproducibility and consistency, the development and evaluation of the proposed compression frameworks were carried out in a structured software and hardware environment. Different environments were used across phases, reflecting the evolution of the research and the specific requirements of each study.

3.4.1 Programming Environment and Libraries:

Python 3.12 served as the primary platform for implementing and benchmarking compression algorithms. Libraries included NumPy, Pandas, Matplotlib, and codec implementations such as zlib, LZ4, Zstandard, LZMA, Brotli, and bzip2. Machine learning modules (Random Forest, Isolation Forest, and Logistic Regression) were implemented using scikit-learn. These libraries ensured coverage of both classical and contemporary codecs.

3.4.2 IDE & Development Environments:

i. **PyCharm Professional (2025 Edition):** Used for Python-based modules, providing debugging, project management, and reproducibility.

ii. **NetBeans 8.2 with JDK 8:** Used during early prototypes of algorithmic modules, especially for ontology-based methods (Papers 2 and 3).

iii. **MATLAB R2015b and MATLAB R2023a:** Used for simulation and validation of semantic and ontological compression models, as well as hybrid frameworks. MATLAB facilitated experimentation with residual and transform-based techniques.

iv. **Protégé 4.3:** Employed for ontology modeling in semantic preprocessing and compression experiments (Papers 2 and 3).

v. **Stanford POS Tagger / Parser:** Applied for part-of-speech tagging in multilingual preprocessing tasks.

vi. **WordNet Lexical Database:** Used as a supporting thesaurus for semantic enrichment and preprocessing in textual and ontology-driven compression.

3.4.3 Hardware Acceleration:

FPGA-related experiments were conducted using Xilinx Vivado 2023.2. The MOR-ALDC compression core was written in VHDL and synthesized for the Xilinx Zynq-7020 (Artix-7 family) platform. RTL simulation validated correctness prior to synthesis and resource utilization analysis.

3.4.4 Edge Testbed and Packet Capture:

For edge-computing experiments, a small IoT testbed with Wi-Fi connectivity was used to capture real-time network traffic. Wireshark served as the packet analyser, ensuring system operation was unaffected during collection. Captured packets represented image, sensor, financial, and textual data, providing realistic IoT workloads for evaluation.

3.5 System Configuration

All experiments were carried out on a standardized computing setup to ensure reproducibility and fair benchmarking across datasets, algorithms, and frameworks. The complete configuration is provided in Table 3.1.

This configuration was consistently used for all experimental phases, unless explicitly stated otherwise in later chapters.

Table 3.1: System Environment and Specifications

Component	Specification
Device Model	Custom Build (Device ID: DAE00FCA-A7AD-46AD-AB687A912ACEAC9D)
Operating System	Windows 11 Home Single Language, Version 24H2 (OS Build 26100.4061)
Processor	13th Gen Intel® Core™ i7-1355U @ 1.70 GHz, 10 Cores / 12 Threads
Memory	16 GB RAM (15.7 GB usable)
Architecture	64-bit OS, x64-based processor
Programming Environment	Python 3.12 with NumPy, Pandas, Matplotlib, zlib, LZ4, Zstandard
IDE	PyCharm Professional 2025 Edition
FPGA Development	Xilinx Vivado 2023.2; VHDL compression core synthesized for Artix-7/Zynq-7020 (Artix-7 family)
Timing Function	Python time.perf_counter() (high-resolution wall-clock timer)
Runs Averaged (N)	5 (arithmetic mean reported; mean ± std where available)
Measured Times	End-to-end, including preprocessing scan and encoding

3.6 Evaluation Metrics

The effectiveness of the proposed compression frameworks was evaluated using a comprehensive set of metrics. These were selected to capture both classical performance measures and resource-aware constraints critical for WSN and IoT deployments. The metrics are defined as follows:

3.6.1 Compression-Oriented Metrics

(i) **Compression Ratio (CR):**
Compression ratio measures the reduction in size achieved by compression. It is defined as:

$$CR = \frac{Size_{original}}{\text{Size}_{\text{compressed}}} \qquad (3.1)$$

where $Size_{original}$ and$\text{Size}_{\text{compressed}}$ denote the original and compressed file sizes, respectively. A higher CR indicates greater compression efficiency.

(ii) **Compression Time (CT):**

The total wall-clock time taken by the algorithm to complete compression, measured in seconds. Reported values include preprocessing and encoding

operations, while excluding delays due to external **Input/Output (I/O)** operations such as network transfer.

(iii) **Compression Speed (CS):**
This metric quantifies how fast the compression algorithm operates:

$$\mathrm{CS} = \frac{\mathrm{Size}_{\mathrm{original}}}{\mathrm{Compression\ Time}} \qquad (3.2)$$

it is expressed in MB/s. Higher CS values indicate faster performance, which is critical for real-time or streaming applications.

(iv) **Decompression Speed (DS):**
Decompression speed evaluates the rate at which a compressed file can be restored to its original state:

$$DS = \frac{\mathrm{Size}_{\mathrm{compressed}}}{\mathrm{Decompression\ Time}} \qquad (3.3)$$

measured in MB/s, a higher DS reflects faster data recovery, essential for time-sensitive environments.

(v) **Compression Percentage (CP):**
Compression Percentage quantifies the proportion of storage space saved after compression, relative to the original size. It is defined as:

$$CP = \frac{\mathrm{Size}_{\mathrm{original}} - \mathrm{Size}_{\mathrm{compressed}}}{\mathrm{Size}_{\mathrm{original}}} \times 100 \qquad (3.4)$$

a higher CP value indicates greater memory savings, reflecting the effectiveness of the compression scheme in reducing storage or transmission requirements.

3.6.2 Error and Quality Metrics

(i) Root Mean Squared Error (RMSE):
RMSE is a standard metric for quantifying reconstruction error between original and reconstructed (or predicted) signals. For compression experiments in IoT and WSN contexts, RMSE is applied to packet-level or data-stream quantities rather than pixel intensities. In this thesis we use packet-level RMSE, computed as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\hat{S}_i - S_i\right)^2} \qquad (3.5)$$

where S_i is the original (true) size of the i-th packet (bytes), $\hat{S}_i$ is the reconstructed (proposed) packet size after decompression (bytes), and NNN is the number of packets in the test set. RMSE is therefore reported in the same units as packet size (bytes). Unless otherwise stated, the aggregate RMSE over the full test set is reported. Note that the subtraction order inside the square does not affect the result:
$(\hat{S}_i - S_i)^2 = (S_i - \hat{S}_i)^2$

(ii) Peak Signal-to-Noise Ratio (PSNR):

Peak Signal-to-Noise Ratio (PSNR) is a widely used metric in image and signal processing to quantify the quality of a reconstructed signal relative to the original. Conventionally, it is defined as:

$$PSNR = 10log_{10}\left(\frac{MAX}{MSE}\right)$$

where MAX denotes the maximum possible pixel or signal value (e.g., 255 for 8-bit data), and MSE is the mean squared error between the original and compressed-reconstructed data. A higher PSNR value indicates better preservation of the original signal, with smaller distortion introduced during compression.

Reformulated PSNR for This Thesis

While the conventional PSNR is tailored for multimedia quality assessment, the focus of this thesis is on lossless and near-lossless compression efficiency in IoT and WSN environments. In such scenarios, the key concern is not pixel-level fidelity but rather the trade-off between data size and reconstruction error. Following our prior work, PSNR is reformulated as:

$$PSNR = 10log_{10}\left(\frac{Original\ File\ Size}{RMSE}\right) \quad (3.6)$$

here, the numerator corresponds to the size of the uncompressed data, while the denominator is the Root Mean Squared Error (RMSE) between the original and reconstructed streams. RMSE is used instead of MSE because it is expressed in the same units as the data, making the ratio more interpretable in compression contexts. This reformulation emphasizes how effectively the compression scheme reduces storage size while keeping reconstruction error minimal an especially relevant measure for resource-constrained IoT and edge platforms.

3.6.3 Energy and Network Metrics

(i) Energy Consumption (E):
For embedded platforms, energy was estimated from processing time

$$E = T \times 0.15 \quad (3.7)$$

where E is in millijoules, Tis runtime in seconds, and 0.15 W is the assumed microcontroller power draw. was critical for IoT and WSN evaluation.

(ii) Energy Savings (ES%):
To quantify transmission efficiency, energy savings were defined as:

$$ES\% = \frac{\mathrm{E_{Original}} - E_{Compressed}}{E_{Original}} \times 100 \quad (3.8)$$

where $E_{Original}$ and $E_{Compressed}$ represent the energy required to transmit the original and compressed bitstreams, respectively.

(iii) Network Lifetime (NL)
Network lifetime is defined as the total operational time until the first sensor node depletes its energy. Mathematically:

$$\mathrm{NL} = \mathrm{T_{first\_dead}} \quad (3.9)$$

where $\mathrm{T_{first\_dead}}$ denotes the elapsed time (or number of operational rounds) until the first node runs out of energy.

(iv) Stability Period (SP)
The stability period is the number of rounds completed before the death of the first sensor node. It is expressed as:

$$\mathrm{SP} = \mathrm{R_{first\_dead}} \quad (3.10)$$

where $\mathrm{R_{first\_dead}}$ is the round index at which the first node becomes non-functional due to energy depletion.

3.6.4 Composite Metrices

i. Efficiency Score (ES)

To capture the overall effectiveness of each algorithm, a composite score was defined. The score integrates CR, CS, and DS through min–max normalization and weighted aggregation:

$$\text{ES}= (0.40\times CR_{norm}) + (0.30\times CS_{norm}) + (0.30\times DS_{norm}) \quad (3.11)$$

where normalization is applied as:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}, \quad X \in \{CR, CS, DS\} \quad (3.12)$$

This ensures fair comparison across heterogeneous metrics. The weights prioritize compression efficiency (40%) while also accounting for speed (30% each for CS and DS).

3.7 Memory Metrics for WSN Implementation

Efficient use of memory is a critical requirement in Wireless Sensor Networks (WSNs) and IoT platforms, where nodes often operate with less than 16 KB of RAM. To capture the memory footprint of the proposed compression schemes, two complementary measures were used.

3.7.1 Peak Working-Set RAM (PWSR):

This represents the highest amount of memory allocated by the compression algorithm during its execution. It was measured using Python's *tracemalloc* tool, which records only the objects created by our code, such as residual buffers, Huffman trees, and bit-packing tables. In practice, this value indicates the minimum amount of RAM a sensor node must reserve to ensure the codec runs without failure.

$$\text{PWSR}_{\text{peak}} = \frac{\text{peak}_{\text{tracemalloc}}}{1024}\,(\text{KB}) \quad (3.13)$$

where $\text{peak}_{\text{tracemalloc}}$ is the maximum allocation reported by tracemalloc (in bytes).

3.7.2 Resident-Set Size Delta (RSS Δ):

This measures the overall growth in memory consumption of the process from

the beginning to the end of the compression task. It was obtained using the *psutil* library, which also accounts for memory taken by the operating system or underlying C libraries. Hence, it represents the worst-case increase in RAM during execution.

$$RSS\Delta = \frac{RSS_{after} - RSS_{before}}{1024} (KB) \qquad (3.14)$$

where RSS_{after} and RSS_{before} RSS_{after} are the memory readings taken at the start and end of execution.

Together, **PWSR** reflects the internal footprint of the algorithm itself, while **RSS Δ** reflects the system-level overhead. Both values were reported in kilobytes, and lower values for each are desirable since they ensure the codec can run on highly resource-constrained devices with sub-16 KB RAM budgets.

The metrics outlined in this section cover all key aspects needed to judge the proposed compression methods—how much data they save, how fast they run, how accurately they reconstruct, how much energy and memory they consume, and how long they can sustain a sensor network. Using this set of measures ensures that results are consistent and comparable across datasets and methods. In the next section, these metrics guide the experimental protocol and provide a fair basis for evaluating the proposed frameworks.

3.8 Implementation Protocol

The proposed methodology was implemented through a structured protocol to ensure that experiments were reproducible, fair, and aligned with the research objectives. This protocol covered the full cycle of dataset preparation, algorithm execution, metric evaluation, and validation, and was applied consistently across all phases of the work.

(i) Dataset Preparation

The datasets described in Section 3.3.2 were first converted into standard input formats. Text corpora were cleaned and tokenized, healthcare logs were organized into hierarchical records, and sensor/IoT streams were divided into fixed-length windows so that different algorithms could be compared on a uniform basis. Synthetic signals were also generated to model-controlled behaviours such as smooth, bursty, and noisy patterns. For the edge-based experiments, packet-level traces were collected using Wireshark and verified through captured screenshots.

(ii) Algorithm Deployment

Reference compressors such as Huffman, RLE, LZW, LZ4, Zstd, LZMA, Brotli, and bzip2 were run using Python or system libraries. The proposed methods including LZWP, OntoRLE, hybrid pipelines, and MOR-ALDC were implemented in Python and MATLAB, with semantic preprocessing supported by Protégé, POS tagging, and WordNet resources.

For hardware validation, the MOR-ALDC codec was described in VHDL and synthesized on a Xilinx Zynq-7020 (Artix-7 family) FPGA using Vivado 2023.2. This allowed comparison between flexible software-based implementations and efficient hardware realizations.

(iii) Metric Measurement

Performance was assessed using the evaluation metrics detailed in Section 3.4. Compression ratio, compression speed, compression time and decompression speed were measured automatically through the Python benchmarking harness. Reconstruction quality was tested using RMSE and the adapted PSNR metric. Energy consumption and savings were estimated using a first-order radio energy model. Memory usage was profiled through two complementary tools: psutil for system-level resident memory (RSS Δ) and tracemalloc for peak working-set RAM (PWSR). For WSN scenarios, network lifetime and stability period were also computed using the LEACH framework.

(iv) Validation and Repetition

To minimize the effect of noise from system load or allocator behavior, each experiment was repeated several times and average values were reported. Cross-dataset testing was carried out to check the generality of each algorithm. For the FPGA implementation, RTL simulation was used to confirm correctness before synthesis, and post-synthesis analysis reported hardware utilization (LUTs, flip-flops, BRAM) and timing performance.

(v) Documentation and Comparison

All results were carefully recorded in structured tables and visualized using graphs for easy comparison. Classical algorithms served as benchmarks, and each proposed method was evaluated in terms of its added benefits—whether in compression ratio, speed, energy efficiency, memory savings, or hardware resource use.

By following this protocol, the experiments remained consistent and transparent across datasets, platforms, and evaluation criteria. It also ensured that both software- and hardware-level implementations were validated under realistic constraints, keeping the methodology tightly connected to the thesis objectives.

3.9 Concluding Remarks

The methodology outlined above provides a structured foundation for evaluating compression techniques across diverse datasets and platforms, ensuring reproducibility and fairness. With the experimental framework, evaluation metrics, and implementation protocol in place, the stage is now set for the first phase of analysis. The following chapter presents baseline benchmarking, where widely used, codecs are evaluated to provide a solid point of comparison for the proposed approaches.

CHAPTER 4

BASELINE ANALYSIS OF COMPRESSION TECHNIQUES

The preceding chapter outlined the methodological framework of this study, including the datasets, evaluation metrics, and experimental procedures adopted. Building on that foundation, this chapter presents a baseline evaluation of widely used lossless compression algorithms. The objective is not to propose new methods, but to establish a clear reference point against which the performance of the techniques developed in later chapters can be measured.

4.1 Introduction

Establishing a performance baseline is a necessary step before moving on to the development of new compression approaches. Lossless compression has been studied for decades, with methods ranging from simple redundancy-removal techniques to advanced modern codecs designed for speed and large-scale systems. Yet, when these algorithms are applied to domains such as Wireless Sensor Networks (WSNs) and the Internet of Things (IoT), their effectiveness is not always clear. These systems operate under strict resource constraints, where compression methods must strike a balance between data reduction and real-time feasibility.

4.2 Research Objectives

The purpose of this chapter is to present a comparative baseline analysis of widely used lossless compression algorithms. Rather than proposing new techniques, this study focuses on how existing methods perform under uniform conditions, drawing attention to the trade-offs between compression efficiency and speed.

This analysis plays three important roles in the thesis:

1) It establishes a reference point for assessing the improvements offered by the algorithms proposed in later chapters.

2) It brings out the strengths and weaknesses of widely used codecs, showing the compromises that exist between compression ratio and compression speed.

3) It reinforces the research gaps identified earlier, illustrating why current approaches often fall short in constrained computing environments.

In summary, this chapter sets the stage for the development work that follows. By examining what existing methods can achieve and where they fall short it provides both the foundation and the motivation for the design of new, more efficient compression techniques.

4.3 Experimental Setup

To establish a fair and reliable baseline, it was important to select algorithms, datasets, and performance measures that represent both the classical foundations and the current state of the art in lossless data compression. The choices were guided by two principles: (i) diversity in algorithmic approach, and (ii) relevance of datasets to the application domains of this thesis, namely text, multilingual content, and sensor data from IoT and WSNs.

4.3.1 Algorithms Considered

Eight representative lossless compression algorithms were selected to cover the spectrum from classical to modern approaches (detailed descriptions are already provided in Chapter 2).

a) **Classical Codecs -** RLE, LZW

b) **Standard / Established Codecs -** DEFLATE (gzip), Bzip2, LZMA

c) **Modern High-Performance Codecs -** Brotli, Zstandard (Zstd), LZ4

This classification ensures that both traditional redundancy-removal methods and contemporary high-performance codecs are fairly benchmarked under uniform conditions.

4.3.2 Benchmarking Datasets

The benchmarking experiments in this chapter were carried out on three categories of datasets, which have already been described in detail in Section 3.2.2. To briefly summarize, plain English text corpora of different sizes (≈165 KB, 1.3 MB, and 10 MB) were chosen to represent structured single-byte character streams.

Alongside these, UTF-8 encoded Hindi text files in the Devanagari script (≈145 KB, 1.6 MB, and 13 MB) were included to capture the added complexity of multi-byte Indic characters and compound glyphs. The third category comprised sensor data, drawn from both real-world and synthetic sources. Real traces were taken from the Intel Berkeley Lab and UCI Air Quality datasets, while synthetic signals with smooth, periodic, bursty, and noisy patterns were generated to emulate typical IoT and WSN conditions. This combination of datasets spans both text and time-series domains, ensuring that the performance benchmarks reflect a broad and representative range of application scenarios.

4.3.3 Evaluation Metrics

For a fair comparison across all algorithms, two standard performance indicators were considered. Compression Ratio (CR) and compression speed (CS) were computed using the definitions already provided in Section 3.6.1 (Equations 3.1 and 3.2). CR reflects the efficiency of size reduction, while compression speed captures processing speed, expressed in megabyte per second (MB/s). Together, these metrics provide a balanced perspective: CR highlights compression efficiency, and demonstrates the practicality of an algorithm in real-time and resource-constrained environments such as IoT and edge computing.

4.4 Performance Results on Benchmark Datasets

To assess codec performance in a fair and representative manner, three categories of data were considered: English text corpora, sensor streams, and Hindi text corpora. Together, these datasets cover both structured and unstructured content, as well as single-byte and multi-byte scripts, providing a balanced basis for comparison. The detailed results for each dataset type are presented in the subsections that follow.

4.4.1 Results on English Text Corpora

Table 4.1 reports the compression ratio and CS achieved by different algorithms on small, medium, and large English text files

Table 4.1: Baseline Compression Performance on English Text Corpora

Compressor	CR (Small / Medium / Large)	CS(MB/s) (Small / Medium / Large)
RLE	0.511 / 0.520 / 0.512	4.30 / 5.07 / 5.19
LZW	2.927 / 3.454 / –	0.22 / 0.05 / –
DEFLATE (gzip)	2.848 / 2.675 / 2.621	11.86 / 9.85 / 10.25
Bzip2	3.615 / 3.429 / 3.493	10.45 / 9.75 / 10.06
LZMA	3.204 / 3.195 / 3.588	2.08 / 1.64 / 0.57
Brotli	2.932 / 2.937 / 3.002	13.00 / 13.59 / 11.66
Zstd	2.923 / 2.877 / 2.967	26.31 / 35.99 / 35.51
LZ4	2.430/ 2.311 / 2.288	15.64 / 9.40/ 12.82

The results in Table 4.1 highlight the contrast between algorithms that maximize compression efficiency and those optimized for speed. Bzip2 consistently achieved the highest ratios, averaging close to 3.6× across all file sizes, but its CS was limited to about 10 MB/s. LZMA also produced strong ratios, but its performance deteriorated with scale, dropping below 1 MB/s on large files, making it suitable only for archival tasks. At the other extreme, LZ4 offered much faster CS in the range of 9–16 MB/s but with lower compression, around 2.3–2.4×, showing that speed was achieved at the cost of efficiency. RLE, as expected, provided no real reduction, and LZW, while initially effective in ratio, collapsed in speed on larger inputs.

Between these extremes, modern codecs provided more balanced trade-offs. Zstandard (Zstd) stood out as the most versatile, delivering ratios near 3× while sustaining CS above 25 MB/s, peaking at 36 MB/s on large files. Brotli and DEFLATE (gzip) provided moderate but reliable results, with ratios of around 2.7–3× and CS between 10–13 MB/s. Overall, the findings show that traditional dictionary and entropy coders are limited in either speed or efficiency, whereas Zstd demonstrates the strongest balance, making it more suitable for environments such as IoT and WSN nodes where both compression and responsiveness are critical.

4.4.2 Results on Sensor Streams

Table 4.2 presents the baseline performance of the selected codecs on three representative sensor data sources: the Air Quality (AirQ) dataset, the Intel WSN

traces, and a synthetic dataset with controlled signal patterns. The results are reported in terms of CR and CS, allowing a comparison of efficiency and speed across different types of sensor streams**.**

Table 4.2: Baseline Compression Performance on Sensor Streams

Compressor	**CR (AirQ / Intel / Synth)**	**CS (MB/s) (AirQ / Intel / Synth)**
RLE	0.562 / 0.523 / 0.541	2.87 / 2.54 / 5.25
LZW	2.221 / 0.902 / 1.976	0.02 / 0.43 / 0.01
DEFLATE (gzip)	2.905 / 1.408 / 2.632	11.14 / 3.36 / 5.25
Bzip2	3.832 / 1.290 / 2.981	12.52 / 1.25 / 3.39
LZMA	3.887 / 1.095 / 2.981	1.94 / 0.02 / 1.62
Brotli	3.338 / 1.586 / 2.665	19.52 / 0.28 / 6.12
Zstd	3.313 / 1.445 / 2.653	49.95 / 1.61 / 24.65
LZ4	2.435 / 0.986 / 2.011	13.65 / 3.64 / 8.04

The benchmarking results on sensor streams show just how much the nature of the data affects compressibility. For the Air Quality dataset, which has relatively structured readings, Bzip2 and LZMA achieved the strongest compression ratios, both around 3.8–3.9×. However, their speeds were very different: Bzip2 reached about 12.5 MB/s, while LZMA slowed to under 2 MB/s, making it less practical for live applications. Zstd offered the best overall balance, with a ratio of 3.3× and by far the highest CS (~50 MB/s), followed by Brotli at 19.5 MB/s. DEFLATE and LZ4 delivered only moderate ratios (≈2.4–2.9×) but ran at 11–13 MB/s, making them more useful where speed matters more than maximum reduction.

On the Intel dataset, which consists of noisier and less regular sensor readings, compression proved much harder. Ratios stayed below 1.6× for most algorithms, with only Brotli and Zstd performing slightly better. CS was also inconsistent: DEFLATE and LZ4 were reasonably fast (≈3–4 MB/s), but Brotli and Zstd slowed sharply, and LZMA dropped to almost unusable levels (~0.02 MB/s). These results underline the fact that highly irregular WSN traces are inherently less compressible, and general-purpose codecs struggle in this domain.

The synthetic dataset, designed with controlled signal patterns, produced more favourable outcomes. Bzip2 and LZMA again reached ratios around 3.0×, but

Zstd proved most effective overall, combining a competitive ratio (2.65×) with the fastest CS (~25 MB/s). Brotli followed with 6 MB/s and ratios near 2.7×, while LZ4 maintained higher speed (~8 MB/s) but compressed less strongly (≈2.0×).

Taken together, these results confirm a recurring trend: traditional codecs such as Bzip2 and LZMA deliver high ratios but are slowed by heavy computation, while modern methods like Zstd and Brotli provide better trade-offs between efficiency and speed. The sharp variation across Air Quality, Intel, and synthetic signals highlights the challenge of identifying a single codec that works well across all sensor data types. This gap underscores the need for domain-aware and hybrid compression approaches, which form the focus of the later phases of this thesis.

4.4.3 Results on Devanagari-Encoded Hindi Text

Table 4.3 reports the performance of the codecs on Hindi text encoded in the Devanagari script (UTF-8). The results, given in terms of Compression Ratio (CR) and CS, illustrate the additional challenges of compressing multi-byte Indic characters compared to single-byte English text.

Table 4.3 Baseline Compression Performance on Hindi Text (UTF-8, Devanagari)

Compressor	CR (Small / Medium / Large)	CS (MB/s) (Small / Medium / Large)
RLE	0.505 / 0.506 / 0.506	7.42 / 7.14 / 3.11
LZW	2.976 / 3.005 / –	0.92 / 0.02 / –
DEFLATE (gzip)	4.511 / 4.477 / 5.391	12.11 / 5.53 / 6.86
Bzip2	6.003 / 7.240 / 8.700	11.87 / 8.52 / 8.81
LZMA	5.021 / 6.501 / 8.425	2.09 / 1.06 / 0.88
Brotli	4.480 / 4.802 / 5.825	15.80 / 13.96 / 15.68
Zstd	4.391 / 4.590 / 5.540	49.40 / 37.94 / 37.41
LZ4	3.594 / 4.116 / 4.923	13.48 / 3.00 / 4.71

The results in Table 4.3 show that Devanagari-encoded Hindi text allows for higher compression ratios compared to English corpora, but CS varies widely. Bzip2 achieved the strongest ratios, rising from 6× on small files to more than 8× on large corpora, though its speed remained modest (8-11 MB/s). LZMA showed similar strength in ratio but became prohibitively slow (<1 MB/s for most cases), making it unsuitable for real-time applications. Classical methods performed inconsistently: LZW compressed effectively on small and medium data but at near-zero speed, while RLE offered virtually no savings. LZ4 also underperformed in this domain, with ratios of only3.5-4.9× and unstable CS.

In contrast, modern codecs offered more practical trade-offs. Brotli maintained ratios near 4.4-5.8× with moderate CS, while Zstandard (Zstd) combined similar ratios with outstanding speed, scaling from 49 MB/s on small files to nearly 37.41 MB/s on large corpora. DEFLATE provided stable but mid-range performance (~4.5× at ~12 MB/s). Taken together, the results confirm that traditional codecs either maximize compression or speed but not both, whereas Zstd offers the most balanced performance for Indic text. These findings strengthen the case for developing hybrid and semantic-aware frameworks, which are explored in later chapters of this thesis.

4.5 Discussion

The baseline evaluation across English text, sensor streams, and Hindi text highlights both recurring patterns and domain-specific challenges. For English corpora, the results were largely as expected. Fig. 4.1 and 4.2 show that traditional codecs such as Bzip2 and LZMA achieved strong compression ratios, but this came at the expense of significantly slower speeds. In contrast, lightweight methods such as LZ4 compressed much faster but provided only modest reductions. Among the modern codecs, Zstandard (Zstd) consistently delivered the most balanced performance, maintaining ratios close to 3× while sustaining very high compression speeds. These findings indicate that while general-purpose text codecs can be effective, their suitability in constrained environments ultimately depends on how they trade off compression efficiency against processing speed.

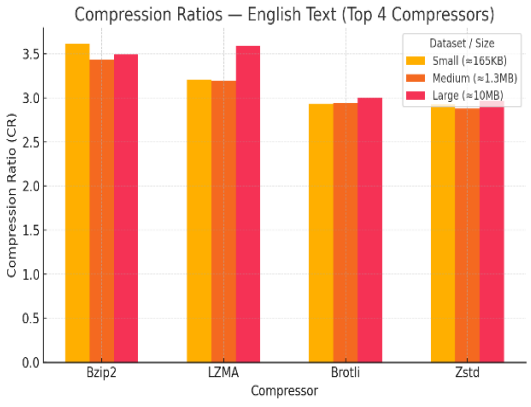

Fig. 4.1: Compression Ratios (CR) of top four compressors across English text corpora (Small, Medium, Large)

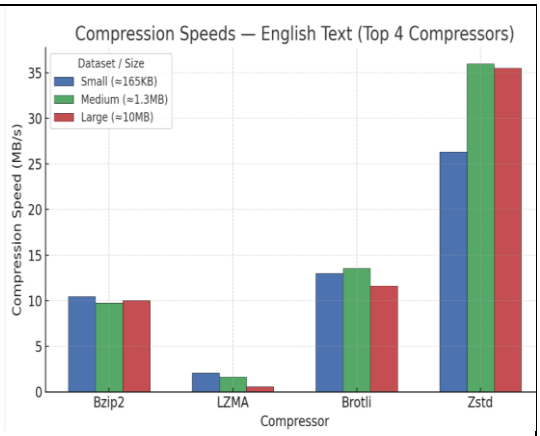

Fig. 4.2: Compression Speeds (CS) of top four compressors across English text corpora (Small, Medium, Large)

For sensor data, the outcomes were more mixed, as illustrated in Fig. 4.3 and 4.4. While structured synthetic signals compressed reasonably well, noisy real-world

traces such as those from the Intel Lab proved difficult to reduce, with most algorithms achieving ratios barely above unity. Even codecs that performed strongly on text, such as Brotli and Zstd, either slowed considerably or lost efficiency on these datasets. This highlights a fundamental challenge: sensor workloads are highly irregular, and general-purpose codecs are often not optimized to cope with the strict energy, memory, and speed constraints of IoT and WSN devices.

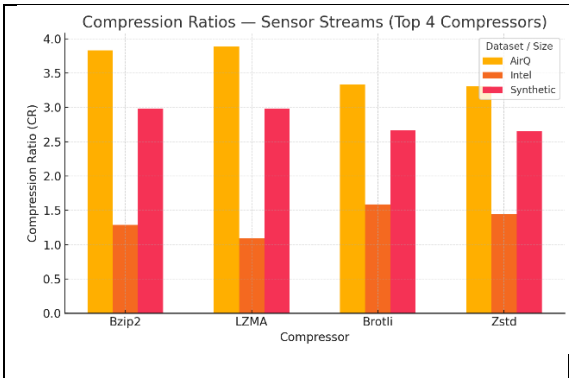

Fig. 4.3: Compression Ratios (CR) of top four compressors on Sensor datasets (AirQ, Intel, Synthetic)

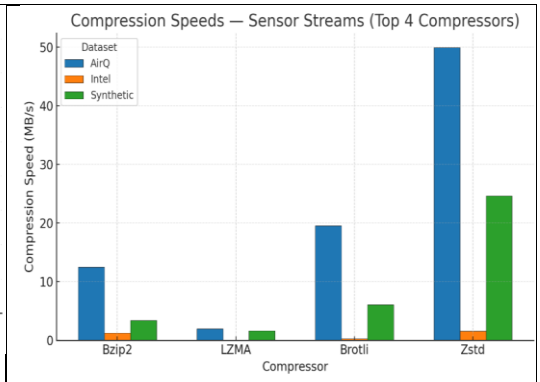

Fig. 4.4: Compression Speeds (CS) of top four compressors on Sensor datasets (AirQ, Intel, Synthetic)

For Hindi text corpora, compression ratios were substantially higher than for English due to the multi-byte nature of the Devanagari script. However, this advantage was offset by inconsistent compression speeds. As shown in Fig. 4.5 and 4.6, Bzip2 and LZMA produced excellent ratios reaching up to 8–9× on large files but were too slow for real-time applications. Similarly, LZW showed inconsistent behavior, working effectively for smaller files but collapsing in performance for larger inputs. Once again, Zstd emerged as the most practical option, combining ratios above 5× with speeds approaching 40 MB/s, demonstrating its adaptability to different linguistic contexts. These observations highlight the need for codecs that can efficiently manage complex scripts without sacrificing responsiveness.

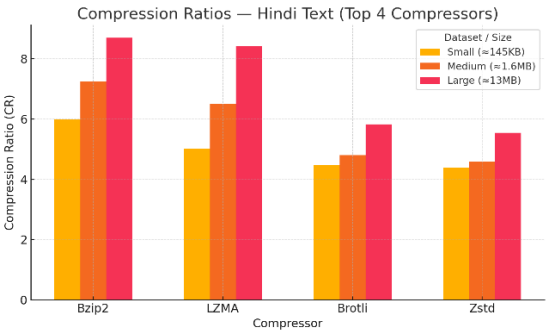

Fig. 4.5: Compression Ratios (CR) of top four compressors across Hindi text corpora (Small, Medium, Large)

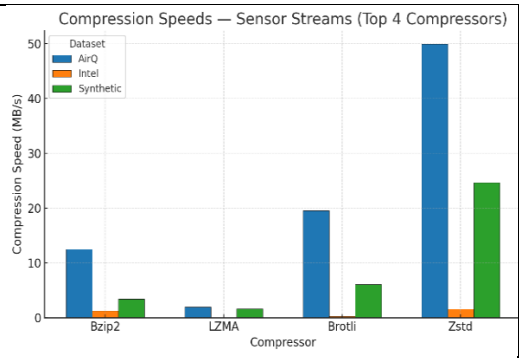

Fig. 4.6: Compression Speeds (CS) of top four compressors across Hindi text corpora (Small, Medium, Large)

Taken together, the findings confirm that no single existing codec performs well across all domains. Standard and classical methods either Favor compression or speed, while modern approaches often rely on large memory buffers and computational power that are not available in low-resource devices. This gap underscores the importance of designing new frameworks that deliberately balance efficiency, speed, and resource usage.

4.6 Concluding Remarks

This baseline study provided a reference point for evaluating widely used lossless compression algorithms across text and sensor data. While the results established clear benchmarks, they also revealed several gaps: the inability to balance speed and compression ratio, the poor adaptability of codecs to non-Latin scripts, and the limitations of existing methods on noisy sensor streams. These shortcomings motivate the advanced approaches developed in the subsequent chapters: optimized dictionary-based methods (Chapter 5), semantic and hybrid pipelines (Chapter 6 and 7), residual and canonical Huffman techniques (Chapter 8), and FPGA-accelerated frameworks for AIoT deployments (Chapter 9). In this way, the experimental gaps identified here directly shape the trajectory of the thesis.

CHAPTER 5

TIME EFFICIENT DICTIONARY BASED COMPRESSION (LZWP)

5.1 Introduction

Dictionary methods are a foundational class of lossless compressors and LZW is among the most widely used due to its simplicity and effectiveness on repetitive inputs. However, in streaming and edge contexts the runtime cost of dictionary maintenance and lookup can make standard LZW impractical. This chapter documents **LZWP** a time-efficient variant of LZW developed as Phase II of this thesis which trades a small one-time preprocessing pass for substantially reduced encoding time in the main compression pass. The method, implementation choices, and experimental evidence shown here directly address the dictionary-overhead bottleneck identified in Chapter 4.

5.2 Research Objectives

The specific objectives of this chapter, corresponding to Phase II of the research methodology, are as follows:

1. Design a lightweight preprocessing strategy that identifies the most frequent byte symbols and builds a compact, high-utility initial dictionary tailored to the input.

2. Integrate the preprocessed dictionary with the standard LZW encoder so that the main encoding pass executes with lower per-symbol overhead while maintaining a comparable compression ratio (CR).

5.3 Experimental Setup -Datasets & Metrics

In this phase, the evaluation of LZWP focused on the English text corpora from the benchmark suite described in Chapter 3. These files represent structured single-byte character streams Using the same text corpora ensures that the improvements of LZWP can be compared directly against the standard LZW implementation under identical conditions. Performance was assessed using the two standard evaluation metrics defined earlier in the thesis: Compression Ratio (CR),

which measures compression efficiency, and compression time (CT)which measures processing time.

The only modification to the setup is that LZWP requires an additional preprocessing pass before encoding. This introduces a buffered two-pass mode, the details of which are explained in Section 5.6.

5.4 Method: LZWP (Preprocessing + Standard LZW)

The standard LZW algorithm begins with a full dictionary of all single-byte symbols and then expands the dictionary dynamically during encoding. While this works well in principle, it also means the compressor repeatedly manages entries for symbols that may never appear, adding unnecessary lookup and update costs. In practice, text files typically exhibit skewed distributions where only a subset of characters occurs frequently.

The key idea in LZWP is to exploit this skew by inserting a short preprocessing step before encoding. A single scan of the input is used to count symbol frequencies, and the initial dictionary is then reordered and trimmed so that it contains only the most frequent symbols. Once this tailored dictionary is in place, the rest of the encoding proceeds exactly like standard LZW. This simple modification significantly reduces lookup overhead while leaving the core mechanics of LZW unchanged.

5.4.1 LZWP Algorithm

The proposed LZWP method consists of a short preprocessing phase followed by standard LZW encoding. In the preprocessing phase, the input is scanned once to record symbol frequencies. Symbols are then sorted by descending frequency, and the most frequent ones (up to 255 entries) are retained to form the initial dictionary. The encoding phase proceeds exactly like LZW, but initialized with this frequency-ordered dictionary, thereby reducing lookup and update overhead.

- **Illustrative Example: “TOBEORNOTTOBE”**

To demonstrate the operation of LZWP, consider the input string **“TOBEORNOTTOBE”**. The distinct characters are {T, O, B, E, R, N}. A frequency scan yields the following counts: O = 4, T = 3, B = 2, E = 2, R = 1, N = 1. Based on this distribution, LZWP initializes its dictionary in frequency order {O, T, B, E, R, N}, whereas standard LZW begins with the same symbols arranged by ASCII order {B, E, N, O, R, T}.

Table 5.1: Initial Dictionary Ordering in LZW (ASCII) and LZWP (Frequency-Based)

Symbol	Frequency	Standard LZW (ASCII order)	LZWP (frequency-ordered)
O	4	4th	1st
T	3	6th	2nd
B	2	1st	3rd
E	2	2nd	4th
N	1	3rd	5th
R	1	5th	6th

Table 5.1 shows that standard LZW initializes symbols strictly in ASCII order: B(66), E(69), N(78), O(79), R(82), T(84). LZWP, in contrast, begins with a preprocessing scan that orders symbols by descending frequency. This ensures frequent symbols such as O and T are prioritized at the front of the dictionary. When two symbols have the same frequency (e.g., B and E, or N and R), ties are resolved by their ASCII codes hence B appears before E, and N (78) appears before R (82).

Table 5.2: Early Encoding Steps on "TOBEORNOTTOBE" under Standard LZW And LZWP

Step	Input fragment	Standard LZW (ASCII order)	LZWP (frequency-ordered)	Dictionary update
1	T	Match late (6th)	Match earlier (2nd)	Add TO
2	O	Match mid (4th)	Match immediate (1st)	Add OB
3	B	Match early (1st)	Match mid (3rd)	Add BE
4	E	Match early (2nd)	Match later (4th)	Add EO

As shown in table 5.2 during encoding, both LZW and LZWP generate identical outputs and extend their dictionaries with the same new entries (TO, OB, BE, EO, …). The only difference is the lookup sequence: in LZWP, frequent symbols such as O and T are found earlier in the dictionary, reducing search time, while in Standard LZW they may appear later due to ASCII ordering. The complete procedure is shown in Algorithm 5.1.

Algorithm 5.1: LZWP — Preprocessing and Encoding

```
# Preprocessing pass
counts = [0] * 256
for b in input_stream:
    counts[b] += 1
symbols_sorted = sort_by_frequency(counts)
initial_dict = symbols_sorted [:DICT_CAP]   # DICT_CAP ≤ 255
# Encoding pass
initialize_dictionary(initial_dict)
encode_with_standard_LZW()
```

5.5 Implementation and Experimental Protocol

The LZWP method was implemented in Python as a software-only modification of a standard LZW encoder. The only algorithmic change is the initial dictionary: symbols are reordered by descending frequency instead of ASCII. Encoding follows standard LZW, so outputs remain decoder-compatible. The implementation is single-threaded and runs in two passes a short frequency scan followed by encoding with the initial dictionary capped at 255 entries and stored in compact array structures to keep lookup overhead low.

Experiments used four English text files of increasing size: L1 (56 KB), L2 (331 KB), L3 (3305 KB) and L4 (9913 KB). Performance is reported using two metrics: CR, which measures size reduction, CSand CT, the end-to-end wall-clock duration of the compress operation.

Timing protocol (brief). Times were measured in Python with a high-resolution wall-clock timer (e.g., time.perf_counter()); each reported value is the arithmetic mean of several independent runs (recommended $N \geq 5$) and includes the

LZWP preprocessing pass. Timings exclude network transfer and external delays beyond local file I/O.

5.6 Results and Discussion

This section presents the comparative performance of standard LZW and the proposed LZWP across the benchmark text files L1–L4. Results are reported in terms of CR ,Total CT andCS. For clarity, tables summarize the numerical values, and graphs illustrate the observed trends. Interpretation is provided alongside each result set.

5.6.1 Compression Ratio Analysis

The CR for each benchmark file was computed using Equation (3.1) defined in Chapter 3. Table 5.3 presents the CR values obtained for the standard LZW and the proposed LZWP methods.

Table 5.3: Compression Ratios of LZW and LZWP on Benchmark Text Files

File	Original Size (KB)	LZW Compressed Size (KB)	LZWP Compressed Size (KB)	LZW Compression Ratio	LZWP Compression Ratio
L1	56	4.00	3.99	14.00	14.04
L2	331	10.00	9.89	33.10	33.47
L3	3305	49.00	48.80	67.45	67.73
L4	9913	102.00	101.40	97.19	97.76

Table 5.3 presents the comparison of compression ratios obtained by standard LZW and the proposed LZWP algorithm across the benchmark text files. The results clearly indicate that both methods achieve nearly identical compression performance, with only marginal differences observable in smaller files. For instance, in L1 and L2, the ratios achieved by LZWP are almost the same as those of LZW, differing only in decimal values that fall within the rounding margin. Similarly, for larger datasets such as L3 and L4, the compression ratios remain practically equivalent, confirming that the frequency-based dictionary initialization of LZWP does not compromise compression efficiency. This consistency across different file sizes demonstrates that the modifications introduced in LZWP primarily target runtime improvements, while the compression effectiveness remains preserved at the same level as the original LZW method. Figure 5.1 plots the same CR values for visual comparison.

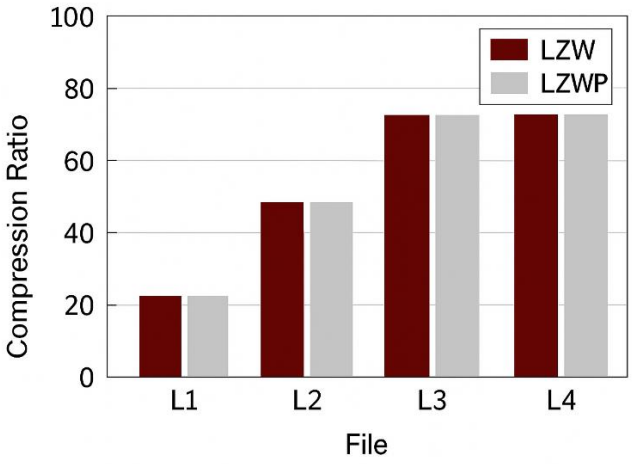

Fig. 5.1: Comparison of Compression Ratios Achieved by LZW And LZWP

5.6.2 Compression Time Analysis

Compression Time (CT) was measured as the total wall-clock duration of the compression process, including preprocessing and encoding steps, as described in Section 3.2. Table 5.4 reports the CT values for both LZW and LZWP across the benchmark text files.

Table 5.4: Compression Times (in seconds) for LZW And LZWP across Benchmark Text Files

Algorithm	L1	L2	L3	L4
LZW	0.7499	3.3018	29.4212	86.7776
LZWP	0.1109	0.4686	4.3053	14.6810

Table 5.4 and Fig. 5.2 present the comparative compression times of LZW and LZWP across the benchmark text files. The results clearly indicate that LZWP delivers faster execution than standard LZW, with the advantage becoming more pronounced as the size of the input grows.

For the smaller files, L1 and L2, LZWP completes compression in 0.11 s and 0.47 s, respectively, whereas LZW requires 0.75 s and 3.30 s. Although the absolute difference is not very large at this scale, it already reflects a six- to sevenfold

improvement in runtime. The impact is far more striking with larger files. On L3, the compression time falls from 29.42 s with LZW to just 4.31 s with LZWP, and for the largest dataset, L4, the difference widens further LZW takes 86.78 s while LZWP finishes in 14.68 s. In both of these cases, LZWP achieves the task in less than one-sixth of the time.

The consistent reduction in execution time can be attributed to the frequency-based dictionary initialization used in LZWP. By prioritizing the most frequent symbols in the initial dictionary, the algorithm reduces lookup overhead during encoding. This optimization ensures that performance gains are not limited to small inputs but scale effectively with larger datasets. Overall, the results confirm that LZWP is a more time-efficient approach, making it especially valuable for medium- to large-scale text compression tasks where runtime efficiency is a key requirement.

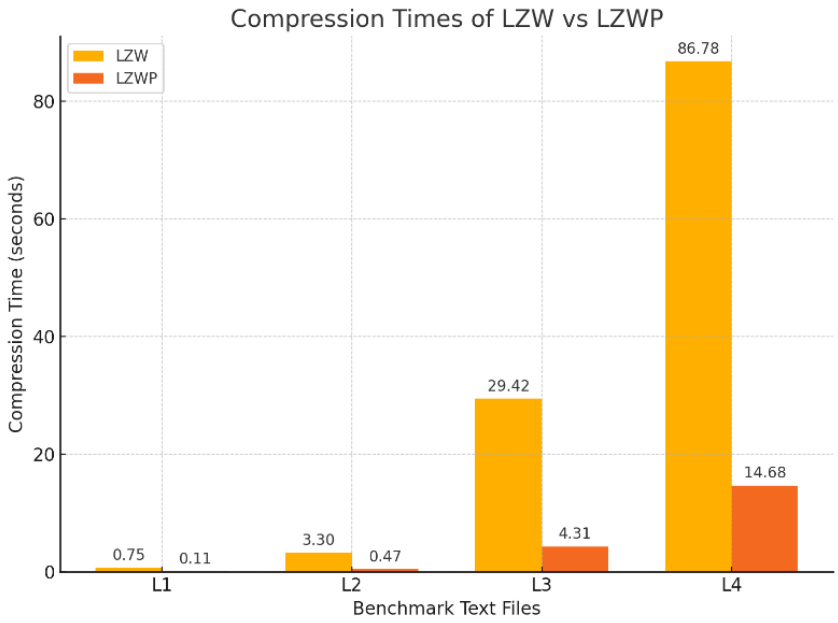

Fig. 5.2: Comparison of Compression Times between Standard LZW and Proposed LZWP

5.6.3 Compression Speed Analysis

Compression Speed (CS) was calculated using Equation (3.2) from Chapter 3, which defines CS as the ratio of the original file size to the corresponding compression time. Table 5.5 lists the CS values for LZW and LZWP, while Figure 5.3 illustrates the comparative performance.

Table 5.5: Compression Speeds (KB/S) of LZW and LZWP across Benchmark Text Files

Metric / File	L1	L2	L3	L4
LZW Speed (KB/s)	74.68	100.25	112.33	114.23
LZWP Speed (KB/s)	504.96	706.36	767.66	675.23

To further evaluate the practical efficiency of the proposed LZWP algorithm, compression speed was computed as the ratio of the original file size to the total compression time. The table 5.3 metric reflects the amount of raw input processed per unit time, thereby providing a clear indication of the throughput achieved during encoding.

The results, presented in Figure 5.3, show that LZWP consistently attains substantially higher compression speeds compared to standard LZW across all benchmark text files. For smaller files such as L1 and L2, the speed-up is already evident, with LZWP processing data at over five times the rate of LZW. As file size increases, the advantage becomes even more pronounced: while LZW reaches a maximum of approximately 114 KB/s on the largest file (L4), LZWP achieves nearly 675 KB/s, representing a sixfold improvement. This confirms that preprocessing based effectively reduces dictionary lookup overhead, allowing the encoder to handle significantly larger input volumes within the same time frame.

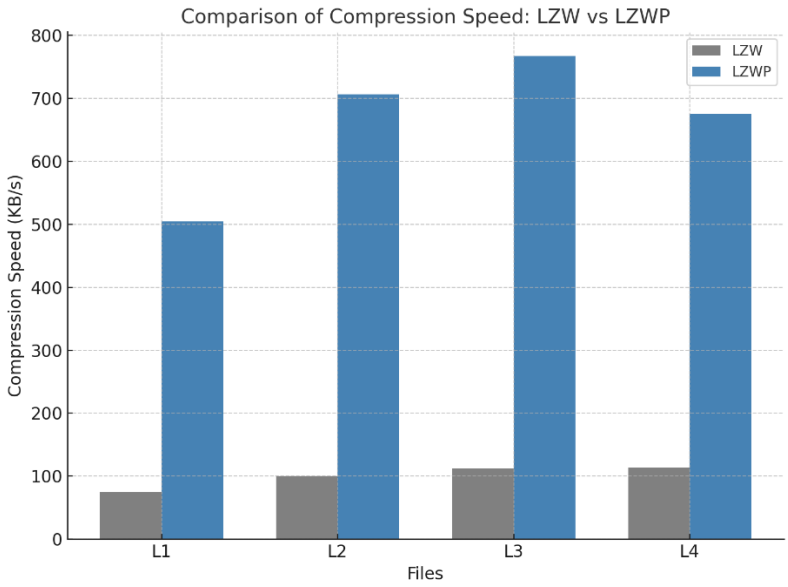

Fig. 5.3: Comparison of Compression Speed Achieved by LZW and LZWP

Together, these results establish that LZWP retains the compression efficiency of LZW while significantly improving execution speed, especially on larger files. The advantage comes entirely from reordering the initial dictionary by

frequency, which reduces lookup cost during encoding. This confirms the effectiveness of the proposed modification and motivates its deployment in settings where runtime is critical.

5.7 Limitations and Applicability

While LZWP demonstrates clear improvements in runtime over standard LZW without compromising compression ratio, certain limitations should be acknowledged. The method requires a two-pass operation: one scan to construct the frequency-based dictionary and another to perform encoding. This restricts its applicability in streaming or real-time environments, where data must be processed sequentially in a single pass. Similarly, the additional preprocessing step provides diminishing returns for very small files, where initialization overhead outweighs runtime savings.

Despite these constraints, LZWP is highly applicable in scenarios where batch compression of medium- to large-scale text files is required and runtime efficiency is critical. Examples include archiving, database export, and server-side log compression, where preprocessing overhead is amortized over large input sizes. By contrast, in resource-constrained IoT and WSN deployments, its two-pass requirement may limit direct applicability, motivating the need for further optimization in subsequent phases of this research.

5.8 Concluding Remarks

This chapter introduced LZWP, a lightweight extension of LZW that addresses the runtime overhead highlighted in Chapter 4. By reordering the initial dictionary based on symbol frequency, the method achieves faster execution without altering the core compression behavior. The contribution of this phase is less about improving compression ratio and more about showing that judicious initialization can make dictionary-based methods practical in time-sensitive contexts. This closes the gap of dictionary maintenance inefficiency and provides a foundation for further exploration.

The next chapters build on this idea by moving beyond dictionary initialization: semantic pipelines (Chapter 6) are explored to handle script sensitivity, while hybrid codecs (Chapter 7) are investigated for more diverse and noisy data such as sensor streams.

CHAPTER 6

SEMANTIC (ONTOLOGY-DRIVEN) COMPRESSION TECHNIQUES

6.1 Introduction

Conventional compressors such as LZW and Huffman operate at the symbol level, focusing only on statistical redundancy. In contrast, IoT and WSN data often contain a high degree of semantic redundancy different readings may carry the same meaning, or values may follow recurring contextual patterns. Symbol-level approaches cannot exploit this higher layer of structure, which limits their effectiveness in application-specific domains.

To address this, this chapter introduces ontology-driven compression, where data is first mapped to domain concepts before encoding. Ontologies capture semantic relationships between values, allowing equivalent or related readings to be clustered and compressed more effectively. Two contributions are presented: an IoT healthcare framework that integrates ontology-based preprocessing with standard compressors, and OntoRLE, an ontology-augmented run-length encoding scheme for WSN data. Together, these methods form Phase 3 of the research methodology, showing how compression can be extended from purely syntactic to context-aware and domain-specific pipelines.

6.2 Research Objectives

This chapter focuses on advancing compression by incorporating ontology-based preprocessing into the pipeline. The objectives are:

1. To develop an ontology-guided healthcare framework that improves compression of IoT medical data by mapping sensor readings to domain concepts before applying standard codecs.

2. To develop OntoRLE, a lightweight extension of run-length encoding that uses ontology-based clustering to enhance compression of WSN data streams.

6.3 Experimental Setup - Datasets & Metrics

To evaluate the ontology-driven approaches in this phase, two datasets were chosen, each reflecting a distinct application domain healthcare and wireless sensor networks (WSNs).

6.3.1 Datasets

i. **Smart Healthcare Dataset (Kaggle):** Used for the IoT-based Smart Healthcare Management framework. It consists of 20 files of different sizes containing patient records, including medical history, camp visits, daily activity logs, and disease monitoring data. Its structured format makes it suitable for ontology-based preprocessing, where raw records can be mapped to higher-level healthcare concepts before compression.

ii. **AGROVOC Multilingual Thesaurus:** Used for the OntoRLE algorithm. Developed by the FAO of the United Nations, AGROVOC contains over 41,000 concepts organized through multilingual and hierarchical relationships. Its large-scale semantic structure provides a strong basis for testing ontology-guided clustering and compression in WSN contexts.

Together, these datasets capture two complementary domains real-world patient data and large-scale semantic dictionaries ensuring a robust evaluation of ontology-aware compression.

6.3.2 Evaluation Metrics

The performance of the proposed methods was assessed using compression, quality, and system-level metrics, selected to reflect the requirements of each domain.

i. For the IoT healthcare framework, evaluation considered standard compression measures: Compression Ratio (CR), Compression Percentage (CP), Compression Speed (CS), and Compression Time (CT). In addition, Root Mean Squared Error (RMSE) and Peak Signal-to-Noise Ratio (PSNR) were used to ensure that compression did not compromise the fidelity of medical data, where accuracy is critical.

ii. For the OntoRLE scheme, evaluation included compression measures (CR, , CP) along with system-level indicators specific to WSNs. Network Lifetime measured how long the network remained fully functional before the first node exhausted its energy, while Stability Period tracked the number of operational rounds before the first node failure. These indicators highlight how semantic compression can contribute to energy efficiency and long-term network sustainability.

6.4 Part A: Ontology-Based Healthcare Framework

The healthcare framework is designed around three interconnected modules: preprocessing, ontology construction, and compression. These modules work together to reduce redundancy in patient records and make the data more structured before applying standard lossless compressors.

6.4.1 Method Overview

1. Preprocessing Module

The first stage prepares raw healthcare records for semantic analysis. Sentences are broken into tokens, high-frequency but semantically irrelevant words (stop words) are removed, and part-of-speech (POS) tagging is applied to identify the grammatical roles of each token. Synonyms are then resolved using WordNet, so that terms with the same meaning (e.g., doctor and physician) are treated uniformly. To strengthen this normalization, semantic similarity between terms is also computed. The framework employs a cosine similarity measure to quantify how closely two terms are related:

$$Sim_{cosine} = \frac{C_{t_1,t_2}}{\sqrt{\sum_{dj} w \times w(t_1,j) . \sum_{dj} w \times w(t_2 j)}} \qquad (6.1)$$

where $C_{t1,t2}$ is the co-occurrence weight showing how often the terms t_1 and t_2 appear together. The values w(t,j) represents the weight of a term t in the j^{th} document. The denominator normalizes the score so that the similarity value always lies between 0 and 1. A value closer to 1 means the terms often appear in similar contexts and are therefore semantically close. This computation is also supported by built-in similarity functions available in tools such as WordNet and Protégé.

The final output of preprocessing is expressed in RDF triples (<subject, predicate, object>), which provide a structured and machine-readable basis for

ontology creation. Figure 6.1 illustrates this step, showing tokenization, POS tagging, and dependency parsing for a sample healthcare sentence.

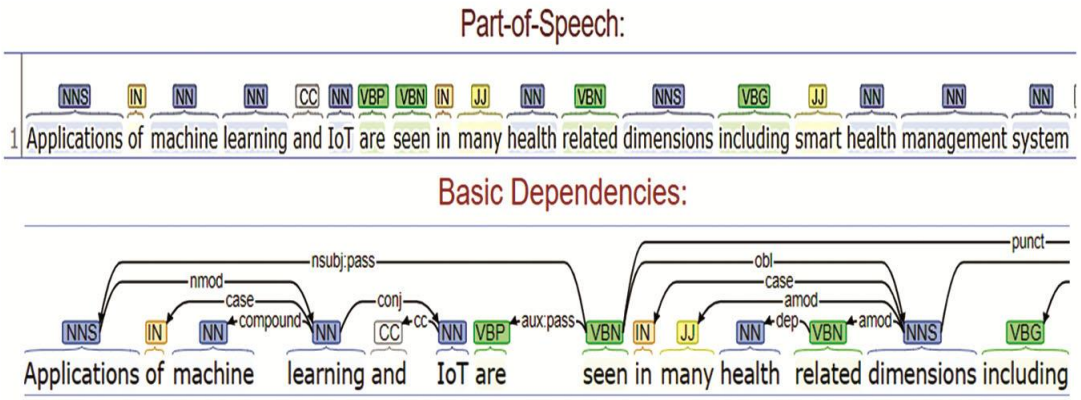

Fig. 6.1: Illustration of Tokenization, POS Tagging, and Dependency Parsing for Healthcare Text Preprocessing

Fig. 6.1 shows how a sample healthcare sentence is processed during the preprocessing stage. The sentence is first broken down into individual words (tokens), each word is tagged with its grammatical role (such as noun, verb, or adjective), and then the relationships between words are identified through dependency parsing. For example, the parser can recognize which word is the subject, which is the verb, and how modifiers like adjectives are connected to nouns. This step transforms unstructured text into a structured form, making it much easier to handle in the next stage where the data is converted into RDF triples and later mapped into the ontology.

2. Ontology Module

The preprocessed terms are organized into a hierarchical knowledge structure using the Protégé ontology editor. Healthcare concepts are represented as classes, properties, and instances, with relationships defined among them (e.g., “Patient” has “Symptom” → “Disease”). By grouping semantically related terms, the ontology reduces variability while preserving meaning, making subsequent compression more effective. Figure 6.2 provides a visual view of a healthcare ontology in Protégé, highlighting how activities and domains are semantically linked.

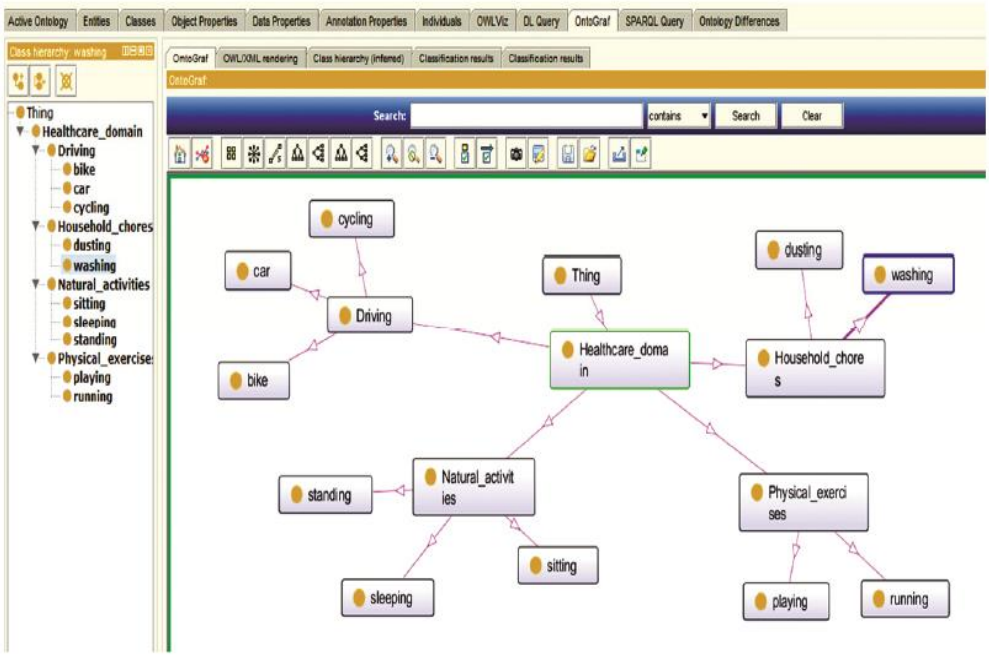

Fig. 6.2: Ontology Graph of Healthcare Concepts Built in Protégé, Showing Hierarchical Relations between Activities and Domains

Fig. 6.2 shows the healthcare ontology as visualized in Protégé using the OntoGraf plugin. The main domain, Healthcare_domain, is divided into categories such as Driving, Household_chores, Natural_activities, and Physical_exercises. Each category is further broken down into specific examples - for instance, Driving includes car, bike, and cycling, while Physical_exercises covers playing and running. This structured view makes it easier to see how different activities are related and grouped under broader healthcare concepts. By organizing data in this way, the ontology reduces confusion and ensures that similar activities are treated consistently, which is especially useful when the data is later compressed.

3. Compression Module

Finally, the semantically normalized dataset is passed to conventional lossless compressors, including LZW, RLE, and Huffman coding. Since the input has been reduced to structured concepts rather than noisy raw text, the compressors achieve better performance. Compression is assessed using Compression Ratio (CR), Compression Speed (CS), and Compression Time (CT). To ensure medical fidelity, RMSE and PSNR are also recorded, confirming that essential healthcare information is preserved. The overall framework is summarized in *Fig. 6.3*, which depicts the flow of data through preprocessing, ontology construction, and compression.

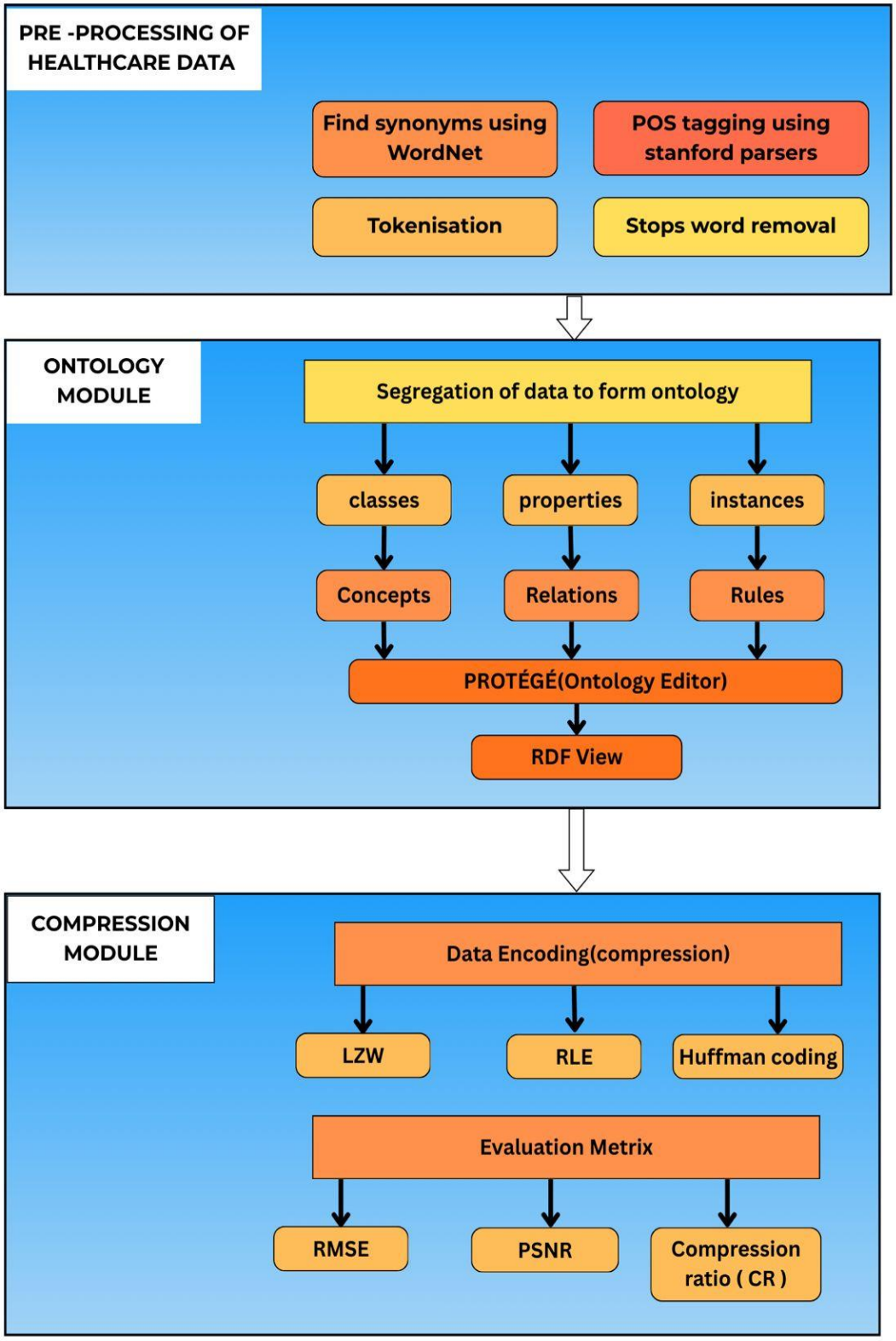

Fig. 6.3: Proposed Ontology-Based Healthcare Framework showing Preprocessing, Ontology Construction, and Compression Modules

Table 6.1: Illustrative Example of the Healthcare Framework

Stage	**Transformation Result**
Raw Input	"The patient visited the camp and was diagnosed with high BP and sugar"
Preprocessing	Tokens → {Patient, visited, camp, diagnosed, high, Blood Pressure, sugar} Stop words removed; abbreviation "BP" expanded to "Blood Pressure"
Ontology Mapping	**Classes**: • Patient → Person • Camp → Event • Blood Pressure, Sugar → Disease **Properties**: • attendedEvent (Person → Event) • hasDisease (Person → Disease) **Instances**: • Blood Pressure (high) → Hypertension • Sugar (high sugar) → Diabetes
Final Structured Form	Patient → attendedEvent → Camp Patient → hasDisease → Hypertension Patient → hasDisease → Diabetes

As shown in Table 6.1, preprocessing cleans and normalizes text, ontology maps terms to formal classes, and compression operates on the structured representation. the raw sentence is first tokenized and stop words are removed, while abbreviations like "BP" are expanded into their full form. This preprocessing yields a set of clean, semantically meaningful tokens. Next, the ontology organizes these tokens into formal concepts: Patient is mapped to the class Person, Camp to Event, and both Blood Pressure and Sugar to the class Disease (with respective instances Hypertension and Diabetes). The final structured record represents only the essential information, eliminating redundant words while retaining meaning. When such normalized records are passed to standard compressors, the resulting data is smaller and easier to store while remaining faithful to the original healthcare information.

6.4.2 Implementation & Experimental Protocol of Healthcare Framework

The healthcare framework was implemented using Java (NetBeans 8.2, JDK) for preprocessing tasks and ontology construction, while ontologies were designed and refined in Protégé 4.3. The compressed outputs were evaluated in MATLAB 2015R13 on a Windows 64-bit platform (Intel i3/i5/i7, ≥2.2 GHz, 4 GB RAM).

For benchmarking, several existing approaches were included alongside the proposed framework. Baseline methods from Pandey et al. (2020), Rasheed et al. (2021), and Sharma et al. (2022) were re-implemented and applied to the same healthcare dataset to allow a fair comparison under identical conditions. These works were selected because they represent recent advances in lossless data compression relevant to structured and healthcare-oriented data. In addition, classical compression schemes such as LZW, Huffman coding, and Run-Length Encoding (RLE) were implemented to provide a reference point against widely used traditional techniques.

Evaluation followed the metrics defined earlier in Section 6.3, namely CR, CP, CS, CT and to confirm the accuracy and integrity of the compressed healthcare records, fidelity indicators such as RMSE and PSNR were also computed. This ensured that comparisons between the proposed method and the baseline algorithms addressed both efficiency and reliability, which are critical requirements in healthcare data management.

6.4.3 Results and Discussion

The proposed ontology-driven healthcare framework was benchmarked against conventional compressors (LZW, Huffman, RLE) and recent semantic-aware studies. The comparative performance is summarized in Table 6.2.

Table 6.2: Comparative Analysis of Ontology-Based Healthcare Framework with Existing Approaches

Approaches / Algorithms	Avg. CR	Avg. RMSE	Avg. PSNR	CT (sec)	CS (MB/s)	CP (%)
Pandey et al.	1.28	1.8295	80.02	6.54	83	21
Rasheed et al.	1.20	1.7767	86.43	6.12	89	16
Sharma et al.	1.23	1.2356	84.45	5.87	87	20
LZW	1.10	1.5567	79.12	5.82	82	23
Huffman Coding	1.18	1.4712	83.76	5.89	76	20
RLE	1.13	1.2583	82.35	6.10	85	25
Proposed	**1.40**	**0.7853**	**91.18**	**5.12**	**91**	**31**

The results bring out several key observations:

The framework achieves the best compression ratio (CR = 1.40), exceeding both classical schemes (LZW: 1.10, Huffman: 1.18, RLE: 1.13) and earlier semantic frameworks (Pandey: 1.28, Rasheed: 1.20, Sharma: 1.23). The improvement stems from semantic preprocessing, which eliminates redundancy before encoding. With the lowest RMSE (0.7853) and highest PSNR (91.18), the proposed approach

preserves medical records more faithfully than other methods (RMSE typically >1.2, PSNR <87). This reliability is critical in healthcare, where diagnostic information must not be distorted. The compression time is reduced to 5.12 seconds, with throughput improved to 91 MB/s. Both values are better than those of the baseline methods (≈5.8–6.1 s, 76–85 MB/s), showing that semantic normalization not only improves efficiency but also accelerates encoding. The compression percentage reaches 31%, the highest across all methods tested. This indicates that semantically reorganized data is inherently more compact, leading to greater storage and transmission savings.

In summary, the ontology-based healthcare framework consistently outperforms both traditional compressors and prior semantic approaches. It provides a strong balance of compression strength, speed, and fidelity, making it a practical solution for IoT healthcare systems where efficiency and accuracy are equally important.

6.4.4 Limitations and Applicability

The healthcare framework benefits from semantic preprocessing, which improves compression by eliminating redundancy and aligning related terms. However, this advantage comes at the cost of additional preprocessing effort. Tasks such as tokenization, POS tagging, synonym resolution, and RDF generation introduce computational overhead, which can be significant for large or streaming datasets. Moreover, the ontology itself requires regular curation by domain experts; without updates, it risks losing clinical relevance. In practice, the framework is best suited to semi-structured repositories such as electronic health records (EHRs) or cloud-based healthcare platforms, where preprocessing time is acceptable and accuracy of stored information is paramount.

6.5 PART B: OntoRLE for WSN Data

OntoRLE adapts ontology-based compression for Wireless Sensor Networks (WSNs), where reducing data size is directly tied to saving memory and extending network lifetime. Instead of treating sensor readings as raw numbers, OntoRLE organizes them into meaningful groups (clusters) using semantic relations, so that only the essential information is stored and transmitted. The workflow of OntoRLE is shown in Fig. 6.4. It begins with a preprocessing step to remove direct repetitions, followed by ontology construction, relation analysis, clustering, reduction, and finally mapping compressed data.

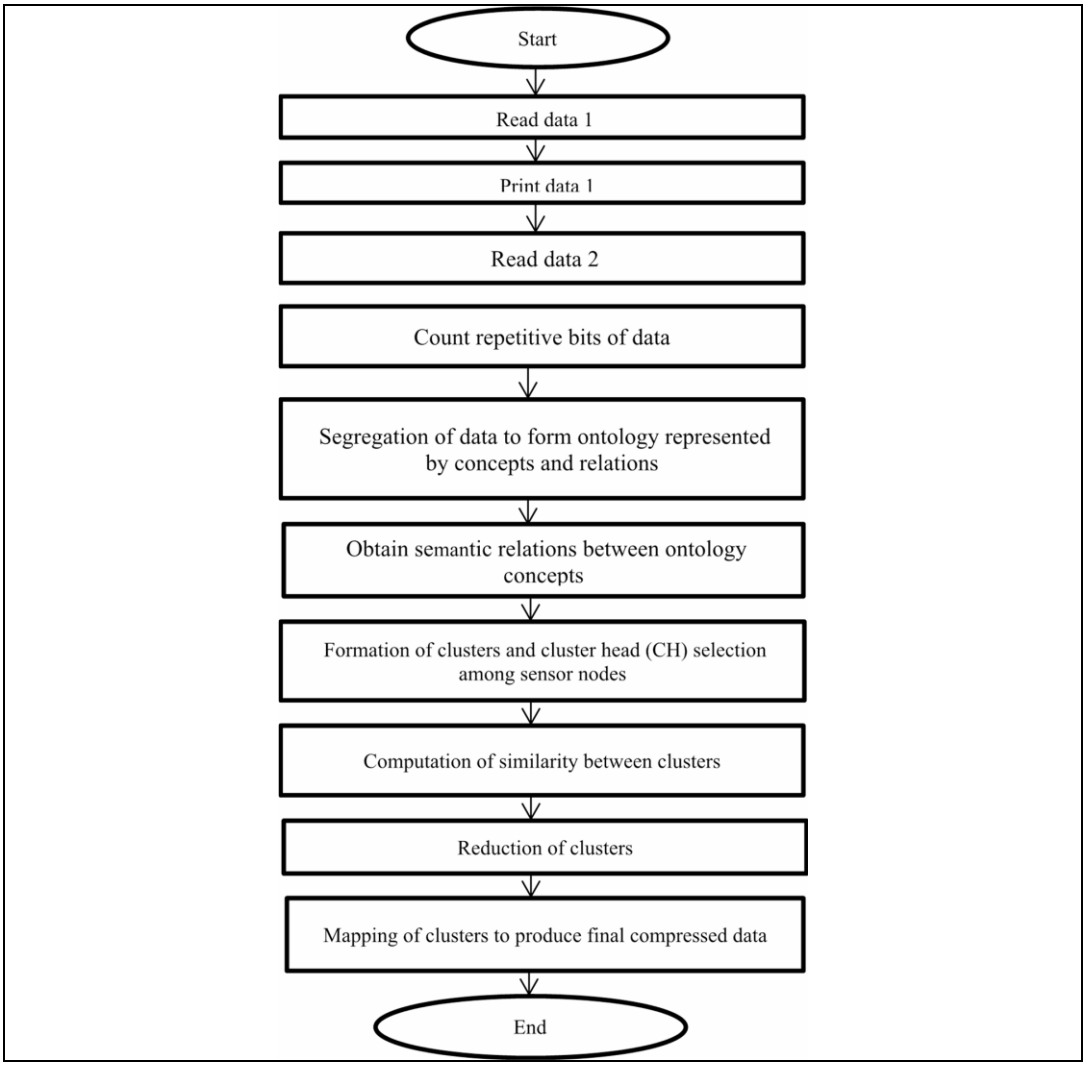

Fig. 6.4: Workflow of OntoRLE

6.5.1 Method Overview

1. **Preprocessing**: Repeated sensor readings are compacted to remove direct redundancy.

2. **Ontology construction**: The cleaned values are then represented as classes, properties, and instances, providing a structured view of the data.

3. **Semantic relation extraction:** Semantic similarity between ontology concepts was computed using cosine similarity, as defined in Eq. (6.1) in Part A. This measure ensures that concepts frequently co-occurring in similar contexts are recognized as semantically related.

4. **Clustering:**
   To extend similarity from individual concepts to groups of related concepts, OntoRLE employs a cluster similarity measure defined as:

$$Sim(G_{1,}G_2) = \frac{\sum(a,b)}{(a+b)} \qquad (6.2)$$

   Here, Sim($G_{1,}G_2$) represents the similarity between two clusters G_1 and G_2. The term Sim(a,b)refers to the pairwise semantic similarity score (from Eq. 6.1) between a concept a in cluster G_1 and a concept b in clusterG_2. The denominator (a+b) indicates the total number of concepts contained in both clusters.

5. **Cluster reduction**: Smaller or weakly related clusters are pruned, reducing dictionary size and memory cost.

6. **Final mapping**: The refined clusters are mapped into compressed records that are compact while still preserving semantic meaning.

- **Illustrative Example**

To illustrate, consider a short set of environmental sensor readings. OntoRLE processes them step by step as shown in Table 6.3.

Table 6.3: Example of OntoRLE Applied to Environmental Sensor Readings

Stage	**Result**
Raw Sensor Input	{Temp: 29°C, Temp: 30°C, Temp: 29°C, Humidity: 65%, Humidity: 65%, Pressure: 101 kPa}
After Preprocessing	Temp {29×2, 30×1}, Humidity {65×2}, Pressure {101×1}
Ontology Mapping	Temp → Class: Temperature; Humidity → Class: Humidity; Pressure → Class: Pressure (all subclasses of Environment)
Semantic Relations	Temp ↔ Humidity (high relation); Pressure (low relation)
Clustering	Cluster 1: {Temperature, Humidity}; Cluster 2: {Pressure}
Cluster Reduction	Pressure cluster removed (weak relation)
Final Mapping	Environment Cluster {Temperature, Humidity} → Compressed record

As shown in Table 6.3, raw sensor data contains repetition (e.g., multiple temperature and humidity readings) and weakly related attributes. OntoRLE first reduces redundancy, then maps each reading into ontology classes: Temperature, Humidity, and Pressure, which are modelled as subclasses of the broader Environment category. Semantic similarity links Temperature and Humidity into a single cluster, while Pressure (less correlated in this example) is pruned. The final compressed record keeps the most meaningful associations while discarding redundant or weakly related information, reducing data size and saving transmission cost in WSNs.

6.5.2 Implementation & Experimental Protocol of OntoRLE for WSNs

OntoRLE was implemented using Java (NetBeans 8.2, JDK) for ontology construction and similarity computations, while MATLAB was employed for clustering, compression, and evaluation of network-level parameters. Experiments were performed on a Windows 64-bit system equipped with an Intel i3/i5/i7 processor (≥2.2 GHz) and 4 GB RAM or higher, consistent with the baseline environment described earlier.

For benchmarking, OntoRLE was evaluated against a diverse set of compression algorithms, covering both traditional methods and more recent hybrid approaches. The baseline group comprised K-RLE (Sharaff et al., 2022), Huffman coding (as applied in WSN contexts by Gupta et al., 2022), LZW (Roy & Chandra, 2020), and RLE (Pushpalatha & Shivaprakasha, 2020), along with transform- and cluster-based techniques such as Discrete Cosine Series (Singh, 2019), Arithmetic Coding (Lungisani et al., 2022), Discrete Wavelet Transform (Manuel et al., 2023), Cluster-based Schemes (Mehta & Lobiyal, 2017), Sparse Reconstruction Algorithm (Li et al., 2019), and the combined Huffman–LZW method (Zheng et al., 2021). These approaches were re-implemented in the same environment and applied to the AGROVOC dataset, ensuring that comparisons with the proposed OntoRLE were fair and consistent across all evaluation metrics.

We evaluated OntoRLE using the metrics defined in Chapter 3 (Section 3.6). Specifically, we report Compression Ratio (Eq. 3.1) and Compression Percentage (Eq. 3.4). To capture network-level effects, we also measured Network Lifetime (Eq. 3.9), and Stability Period (Eq. 3.10). All metrics follow the units and measurement protocol set out in Chapter 3 to ensure direct comparability across experiments. Together, these measures capture not only compression efficiency but also the impact of semantic compression on the sustainability of WSNs, bridging storage performance with energy-aware evaluation.

6.5.3 Results and Discussion

The evaluation of OntoRLE was carried out against several well-known compression techniques, including K-RLE, HC, LZW, RLE, DCS, AC, DWT, CS, SRA, and the hybrid HC+LZW approach. The comparison employed four key performance indicators: CR, CP, NL, and SP, as summarized in Table 6.4.

Table 6.4: Comparative Analysis of the Proposed OntoRLE with Existing Recent Studies

Techniques / Algorithms	CR	Network lifetime (%)	Stability period (%)	CP (%)
K-RLE	1.13	27.67	20.11	23.11
HC	1.16	26.34	13.1	14.77
LZW	1.13	22.1	12.13	18.33
RLE	1.16	24.1	11.1	24.12
DCS	1.13	23.66	10.12	27.33
AC	1.17	28.55	12.67	13.45
DWT	1.13	22.35	10.55	11.34
CS	1.16	27.54	14.78	16.57
SRA	1.16	26.34	13.1	13.44
HC + LZW	1.16	29.43	18.34	25.76
OntoRLE	1.22	32.87	24.11	30.67

In terms of compression ratio, OntoRLE consistently achieved higher values than competing schemes. While existing algorithms demonstrated CR values in the narrow range of 1.13–1.17, OntoRLE attained a ratio of 1.22. This trend is clearly illustrated in Fig. 6.5, where OntoRLE stands out above all baseline methods. Although the absolute improvement appears modest, such increments are significant in wireless sensor network (WSN) environments, where even small gains in compression translate into reduced storage, faster transmission, and measurable energy savings over extended operation.

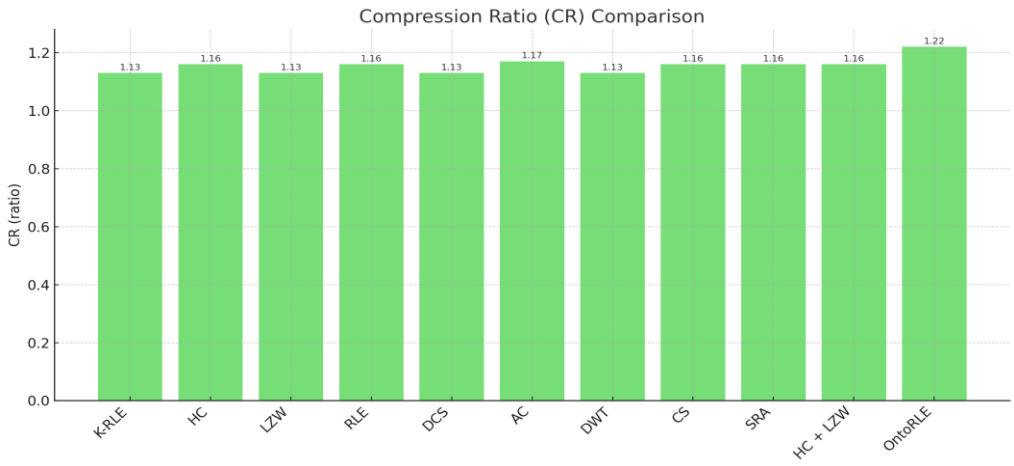

Fig. 6.5: Compression Ratio comparison of OntoRLE with existing algorithms

The advantage of OntoRLE becomes more evident when viewed through compression percentage. Competing methods produced CP values ranging from 11.34% to 27.33%. OntoRLE achieved a markedly higher CP of 30.67%, reflecting its ability to eliminate redundancy through semantic grouping of sensor readings. This improvement is visualized alongside the network-specific metrics in Fig. 6.6, highlighting OntoRLE's superior compression effectiveness compared to conventional and hybrid approaches.

The network lifetime results further demonstrate the impact of OntoRLE. While conventional schemes such as AC and HC+LZW extended lifetime to 28.55% and 29.43%, respectively, OntoRLE pushed this figure to 32.87%. The increase of more than three percentage points over the best existing method, as shown in Table 6.4 and Fig. 6.6, highlights the direct benefit of semantic-aware compression in conserving node energy, thereby prolonging network functionality.

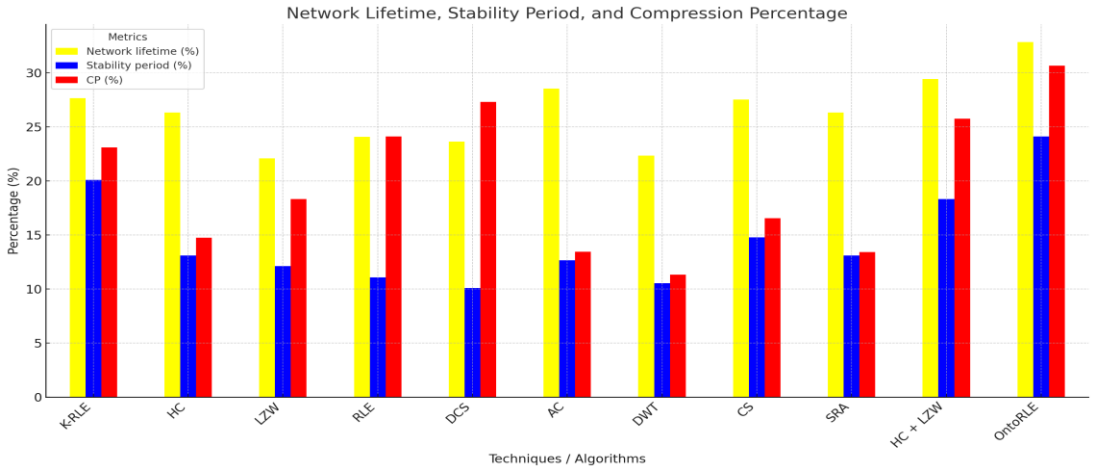

Fig. 6.6: Comparison of OntoRLE with existing Algorithms in terms of Network Lifetime, Stability Period, and Compression Percentage.

A similar trend is observed in the stability period. Existing approaches generally fell between 10% and 18%, indicating earlier node failures. OntoRLE extended stability to 24.11%, which means that a larger proportion of sensor nodes remained alive and active for longer durations. This improvement strengthens the reliability of the network and ensures a more balanced energy consumption across nodes.

Taken together, the results confirm that OntoRLE outperforms classical, transform-based, and hybrid algorithms across all evaluation metrics. Its ability to integrate ontological reasoning with lightweight compression not only improves compression efficiency but also translates directly into prolonged network lifetime and enhanced stability. This validates the premise that embedding semantic awareness into compression frameworks yields tangible advantages for energy efficiency and memory utilization in 5G-enabled WSNs.

6.5.4 Limitations and Applicability

OntoRLE improves compression and extends network lifetime by grouping semantically related sensor readings before applying run-length encoding. Yet, building and managing such ontologies requires additional processing, which may be too demanding for highly resource-constrained sensor nodes. Similarity calculations and cluster management can also introduce delays, making real-time use challenging unless supported by gateways or edge devices. The method is therefore most applicable in scenarios where energy efficiency and network longevity matter more than strict latency such as long-term environmental monitoring, smart agriculture, or structural health sensing.

6.6 Conclusion

This chapter demonstrated the role of semantics in strengthening compression. In healthcare records, ontology-based preprocessing normalized text into structured RDF triples, enabling classical compressors to operate with higher efficiency while preserving accuracy. In WSNs, OntoRLE integrated ontology-driven clustering with RLE to achieve superior compression and extend network lifetime, addressing the dual constraints of memory and energy.

Overall, the results establish that ontology-aware pipelines outperform purely statistical methods by embedding domain knowledge into the compression process. However, the computational cost of ontology construction and the difficulty of adapting these methods to real-time systems remain important challenges. These observations naturally lead to the next step of this thesis: Hybrid Lossless Compression Techniques, where semantic and statistical methods are combined to balance efficiency, speed, and adaptability across heterogeneous data domains.

CHAPTER 7

Hybrid Lossless Compression Techniques

In the previous chapter, we have discussed ontology-based methods for meaning-based data compression. In this chapter we move to a new area compressing Indic scripts, mainly Hindi text written in Devanagari.

7.1 Introduction

Most existing compression methods were created for English and other Latin-script languages. But Indian scripts like Devanagari are different. Each character may have extra marks, joined letters, or compound forms, which make the data larger and harder to compress with standard tools. With the rise of Hindi digital content in education, e-governance, media, and online platforms, there is a clear need for compression methods that are fast and efficient. Independent algorithms often find it difficult to maintain a balance between CR and CS. To overcome this limitation, hybrid lossless compression techniques are used. In a hybrid cascade, two algorithms work one after the other, the first handles strong compression, while the second focuses on improving speed. This combination leads to better and more stable performance, especially while dealing with complex text data such as Indic scripts.

2.11 Research Objectives

The main goals of this chapter are:

1. To create and test hybrid lossless compression methods that combine two algorithms for better performance on Hindi text written in the Devanagari script.
2. To study the balance between Compression Ratio (CR), Compression Speed (CS), and Decompression Speed (DS), and to find the most practical hybrid method that gives the best results for real-world use.

7.3 Experimental Setup — Datasets & Metrics

The hybrid compression methods were tested on Hindi text written in the Devanagari script. Three datasets were used small, medium, and large to check how the methods perform with different amounts of data. These sizes represent real-world cases such as short records, reports, and large digital text files.

Both single and hybrid methods were checked using the same measures CR, CS, and DS. To make the comparison fair, we also used one combined score called Efficiency Score (ES). In this score, 40% importance was given to the compression ratio, and 30 % to compression and decompression speeds each. The basic idea of this scoring method was already discussed in Chapter 3.

7.4 Method Overview

Hybrid compression makes use of the strong points of two algorithms, one after another. This two-step way helps to keep balance between compression ratio and compression speed. It works quite well for Indic text, especially for Devanagari, where many letters get joined or carry extra marks. Such patterns are bit complex, and single algorithms mostly fail to handle them properly.

For testing, each algorithm was used with its normal default settings as shown below:

i. Zstandard (Zstd) – Level 6, no dictionary.

ii. Bzip2 – Default setup, block size 0.9 MB.

iii. LZMA – Compression level 6, no dictionary.

iv. LZ4HC – High-compression mode, level 6, no dictionary.

v. Brotli – Quality level 6, window size 22.

All single algorithms were tested on their own to record the base results. After that, every possible hybrid pair was created by running one algorithm after another, such as Zstd → LZ4HC, Brotli → Bzip2, and LZMA → Zstd. In total, twenty such combinations were formed by pairing each of the five base algorithms with the other four. Table 7.1 lists all these pairs.

Table 7.1: Hybrid Compression Pipeline Matrix

	LZMA	**Zstd**	**Brotli**	**Bzip2**	**LZ4HC**
LZMA	–	LZMA → Zstd	LZMA → Brotli	LZMA → Bzip2	LZMA → LZ4HC
Zstd	Zstd → LZMA	–	Zstd → Brotli	Zstd → Bzip2	Zstd → LZ4HC
Brotli	Brotli → LZMA	Brotli → Zstd	–	Brotli → Bzip2	Brotli → LZ4HC
Bzip2	Bzip2 → LZMA	Bzip2 → Zstd	Bzip2 → Brotli	–	Bzip2 → LZ4HC
LZ4HC	LZ4HC → LZMA	LZ4HC → Zstd	LZ4HC → Brotli	LZ4HC → Bzip2	–

With these settings, the five independent algorithms were tested on small, medium, and large datasets, giving a total of 15 basic test cases. The 20 hybrid pairs were also checked on the same three datasets, adding up to 60 more cases. Together, this made 75 different tests. The results were compared using four measures (CR), (CS), (DS), and (ES). This testing process helped us identify the best hybrid methods based on real results.

Algorithm 7.1 shows the step-by-step flow of the hybrid compression and decompression process. It describes how the two cascaded algorithms work one after the other during compression and in the opposite order during decompression. It also notes the time taken and the data speed for each run.

Algorithm 7.1: Pseudocode for the Proposed Hybrid Compression–Decompression Pipeline

```
// --- Compression ---
   Read input text from InputFile
   OriginalSize ← Size of InputFile
   Step 1: Start timer T1
   Step 2: Compress using Compressor1 → Compressed1
   Step 3: Compress Compressed1 using Compressor2 → FinalCompressed
   Step 4: Stop timer T2
   CompressedSize ← Size of FinalCompressed
   Compression Time ← T2 - T1
   Compression Ratio ← OriginalSize / CompressedSize
   Compression Speed ← CompressedSize / Compression Time  // in MB/s
   Save FinalCompressed
   // --- Decompression ---
   Step 5: Start timer T3
   Step 6: Decompress FinalCompressed using Compressor2 → Partial
   Step 7: Decompress Partial using Compressor1 → Reconstructed Text
   Step 8: Stop timer T4
   Decompression Time ← T4 - T3
   Decompression Speed ← OriginalSize / Decompression Time  // in MB/s
   Output Reconstructed Text
End Algorithm Algorithm Hybrid Compression Decompression(InputFile,
Compressor1, Compressor2)
```

7.5 Implementation and Experimental Protocol

All experiments were done on a laptop with an AMD Ryzen 9 5900HX processor (3.30 GHz) and 16 GB RAM, running on Windows 11 Home (64-bit). The programs were written in Python 3.11 and tested in the IDLE environment to keep the process simple and uniform. The hybrid compression methods explained in Section 7.4 were tested as two-step setups. In this setup, data was first compressed using two algorithm one after other and then decompressed in the reverse order. Both single and hybrid methods were tested on the three datasets mentioned earlier

To get the more reliable results, we run each algorithm three times, and the average values of compressed size, compression time, and decompression time were taken for comparison. The results were measured using the same four metrics CR, CS, DS and ES. All results were shown in tables and charts for easy comparison between the methods and dataset sizes. The code and datasets were saved carefully so that the tests can be checked or repeated later. This setup gave a clear and fair base to study how each compression method worked in real conditions.

7.6 Results and Discussion

This section shows how the single and hybrid compression methods performed on Hindi text written in the Devanagari script. The results are explained here based on the research objectives outlined in Chapter 1 and the research gaps identified in Chapter 2.

7.6.1 Compression Ratio Analysis

Table 7.2: Top Performing Algorithms Based on Compression Ratio

Data Size	**Standard Algorithm (CR)**	**Hybrid Algorithm (CR)**
Large (LD)	Brotli (117.11)	LZMA + Brotli (141.59)
Medium (MD)	Bzip2 (7.56)	Bzip2 + Brotli (7.60)
Small (SD)	Bzip2 (6.57)	Bzip2 + Zstd (6.57)

From the Table 7.2 we can notice that for the small dataset, the Bzip2 algorithm gave the best compression ratio of 6.57. Its hybrid versions, Bzip2+Brotli and Bzip2+Zstd, showed almost the same results. For the medium dataset, a small improvement was seen when Bzip2 was used with Brotli, giving a ratio of 7.60, compared to 7.56 for Bzip2 alone.

From the data, we can see that the biggest gain came with the large dataset. When we look at Table 7.3, we can see that Zstd worked the fastest among all single algorithms. In the small dataset, the hybrid Zstd with LZ4HC gives the best result with speed came around 46 MB/s, while Zstd alone gave almost the same result. For the medium dataset too, Zstd stayed on top, near 77 MB/s, and the other hybrid setups were close in speed.

For the large dataset, the speed improved quite a lot. The pair of Zstd and Brotli worked much faster and reached nearly 1070 MB/s. Zstd alone was almost half of that. So, we can say that using two methods together really helps for bigger text files. The mix of Zstd's fast working and Brotli's stronger compression makes this pair very handy, especially for real-time or cloud-based systems where both speed and space are important.

7.6.2 Compression Speed Analysis

Table 7.3: Top Performing Algorithms Based on Compression Speed

Data Size	Standard Algorithm (CS)	Hybrid Algorithm (CS)
Large (LD)	Zstandard (546.22)	Zstandard + Brotli (1071.57)
Medium (MD)	Zstandard (76.61)	Zstandard + Brotli (70.34)
Small (SD)	Zstandard (45.99)	Zstandard + LZ4HC (46.31)

When we look at Table 7.3, we can see that Zstd worked the fastest among all single algorithms. In the small dataset, the hybrid Zstd with LZ4HC was a little faster. Its speed came around 46 MB/s, while Zstd alone gave almost the same result. For the medium dataset too, Zstd stayed on top, near 77 MB/s, and the other hybrid setups were close in speed.

For the large dataset, the speed went up quite a lot. The pair of Zstd and Brotli worked much faster and reached nearly 1070 MB/s. Zstd alone was almost half of that. So, we can say that using two methods together really helps for bigger text files. The mix of Zstd's fast working and Brotli's stronger compression makes this pair very handy, especially for real-time or cloud-based systems where both speed and space are important.

Table 7.4: Top Performing Algorithms based on Decompression Speed

Data Size	Standard Algorithm (DS)	Hybrid Algorithm (DS)
Large (LD)	Zstandard (593.79)	Zstandard + LZ4HC (663.78)
Medium (MD)	LZMA (456.98)	LZ4HC + Zstandard (485.04)
Small (SD)	LZMA (281.19)	LZ4HC + Zstandard (323.83)

7.6.3 Decompression Speed Analysis

As seen in Table 7.4, the hybrid methods worked better during decompression too. For the small and medium datasets, the LZ4HC+Zstd mix reached speeds of 323.83 MB/s and 485.04 MB/s. This was about 15% and 6% faster than the best independent algorithm (LZMA).

For the large dataset, the Zstd+LZ4HC hybrid gave the highest speed of 663.78 MB/s, which is around 12% faster than Independent Zstd. From the data it is confirmed that hybrid setups can read data faster without losing compression quality which are helpful in systems where data must be accessed quickly, like cloud storage or real-time platforms.

7.6.4 Efficiency Score Comparison

Table 7.5: Normalized Efficiency Scores for Small, Medium, and Large Datasets

Data Size	Standard Algorithm (ES)	Hybrid Algorithm (ES)
Large (LD)	Zstandard (0.683)	Zstandard + LZ4HC (0.860)
Medium (MD)	Zstandard (0.629)	Zstandard + Brotli (0.580)
Small (SD)	Zstandard (0.540)	Zstandard + LZ4HC (0.680)

In Table 7.5, the efficiency score (ES) combines the results of CR, CS, and DS. Among all the methods tested, the Zstd+LZ4HC hybrid gave steady and reliable outcomes. Its ES value was 0.676 for the small dataset and 0.860 for the large one.

For the medium dataset, Zstd alone had the highest score of 0.629, but the difference from the hybrid result was very small. On the large dataset, the Zstd+LZ4HC mix performed about 25% better than Zstd alone. This suggests that hybrid methods can maintain a good balance between speed and compression, especially when the data size increases.

7.6.5 Statistical Analysis and Performance Trends

From Table 7.6 and Figure 7.1, it’s clear that hybrid methods worked better than Independent. The gap between them grew even more when the dataset size increased. Among all, the Zstd based combinations like Zstd+LZ4HC and Zstd+Brotli gave the most balanced and reliable results. This shows that Zstd acts as a strong base for building hybrid compression setups. The ES score also highlights that when the data size becomes large, the DS starts to matter more. This is especially true for real-world uses like mobile apps and digital libraries.

Table 7.6: Comparative Efficiency of Top 10 Compression Methods

Rank	Algorithm (SD)	Efficiency (SD)	Algorithm (MD)	Efficiency (MD)	Algorithm (LD)	Efficiency (LD)
1	Zstd+LZ4HC	0.676	Zstd (Indep.)	0.629	Zstd+LZ4HC	0.860
2	LZ4HC+Zstd	0.571	Zstd+Brotli	0.582	Zstd+Brotli	0.806
3	Bzip2+LZ4HC	0.559	Zstd+LZ4HC	0.568	Zstd (Indep.)	0.684
4	Bzip2+Zstd	0.556	Bzip2+Zstd	0.491	Zstd+Bzip2	0.678
5	Bzip2+Brotli	0.548	Bzip2+Brotli	0.491	Brotli+Zstd	0.668
6	Brotli+Zstd	0.543	Bzip2+LZ4HC	0.490	Zstd+LZMA	0.654
7	Zstd (Indep.)	0.541	LZ4HC+Zstd	0.482	Brotli+LZ4HC	0.612
8	Zstd+Brotli	0.495	Bzip2 (Indep.)	0.472	Brotli (Indep.)	0.607
9	LZMA (Indep.)	0.484	Bzip2+LZMA	0.458	Brotli+LZMA	0.582
10	Bzip2+LZMA	0.479	Brotli+Zstd	0.462	Brotli+Bzip2	0.573

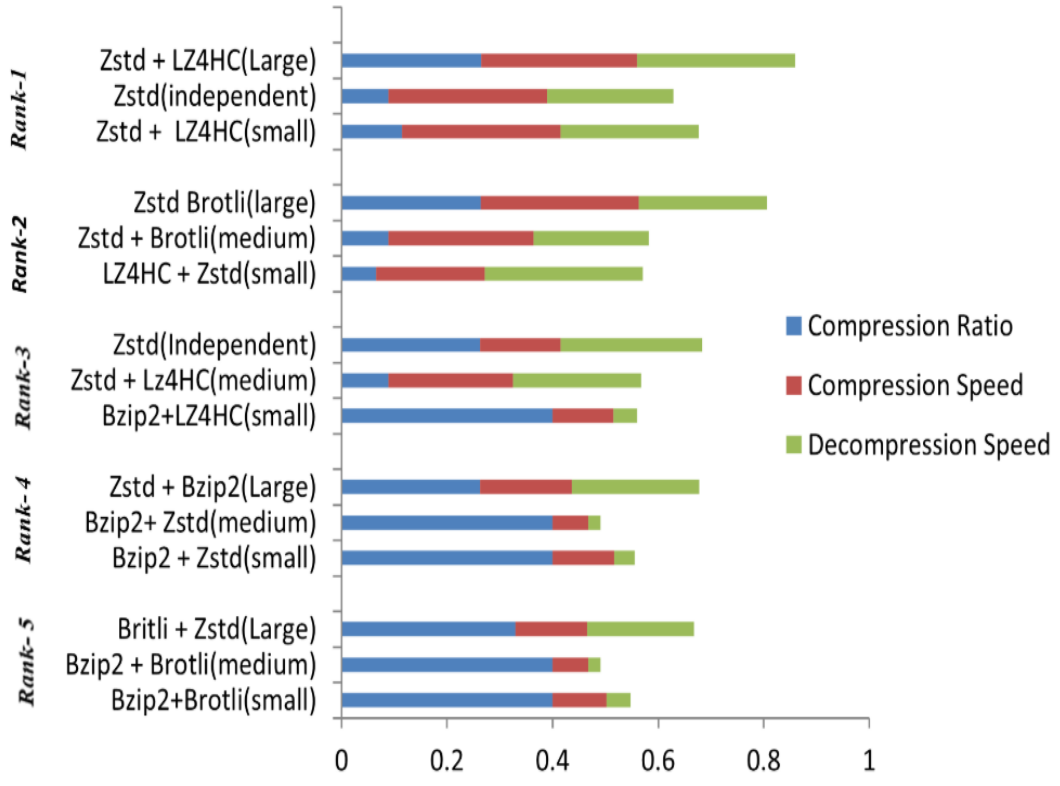

Fig. 7.1: Efficiency Score Trends Across Dataset Sizes

Table 7.7: Comparative Performance of Proposed Hybrid vs. Standard Algorithms

Algorithm	CR	CS (MB/s)	DS (MB/s)	ES
Zstd+LZ4HC	94.82	1055.69	663.78	0.8597
Zstd	94.49	546.22	593.79	0.6836
LZ4HC	3.79	32.51	554.34	0.2580
Bzip2	9.77	18.75	4.88	0.0210
LZMA	141.91	5.96	16.56	0.0573
Brotli	117.11	312.66	426.47	0.6073

7.6.6 Comparative Analysis of Proposed Hybrid vs. Standard Algorithms

If we look at Table 7.7, the hybrid setup Zstd+LZ4HC clearly worked better than the other methods tested. It managed to get an overall efficiency score of 0.8597, which is quite high. The reason is simple it handled both speed and compression well. During testing, it reached about 94.82 in compression ratio, and its speed stayed high in both directions, with over 1050 MB/s for compression and 660 MB/s for decompression.

On the other hand, the Independent Zstd method gave a lower score of 0.6836. Some others, like LZMA and LZ4HC, were strong in only one area but could not balance both speed and compression. So overall, we can see that Zstd+LZ4HC stands out as a good and balanced option that meets the goal of creating a fast and efficient hybrid method.

7.6.7 Trade-off Analysis of Compression Ratio and Speed

Table 7.8: Trade-off Between Compression Ratio and Compression Speed on Large Dataset

Algorithm	CS (MB/s)	CR
LZMA + Brotli	6.09	141.59
LZMA + LZ4HC	6.01	141.10
Brotli + Zstd	491.50	117.10
Brotli + LZ4HC	426.05	117.08
Brotli + LZMA	208.44	117.50
Zstd + LZ4HC	1055.69	94.82
Zstd (Independent)	546.22	94.59
Zstd + Brotli	1071.57	94.49

When we look at Table 7.8 and Figure 7.2, it becomes easy to notice that how the CR and CS work against each other. Some methods gave very high compression but took a lot of time, while others were faster but with smaller savings in space. For example, the LZMA-based hybrids like LZMA+Brotli and LZMA+LZ4HC gave excellent compression, crossing 141, but they were quite slow, running near 6 MB/s make them suitable for long-term storage. On the other hand, Zstd+Brotli and Zstd+LZ4HC worked much faster, going beyond 1000 MB/s, though their ratios were around 94.5 these are better for real-time systems.

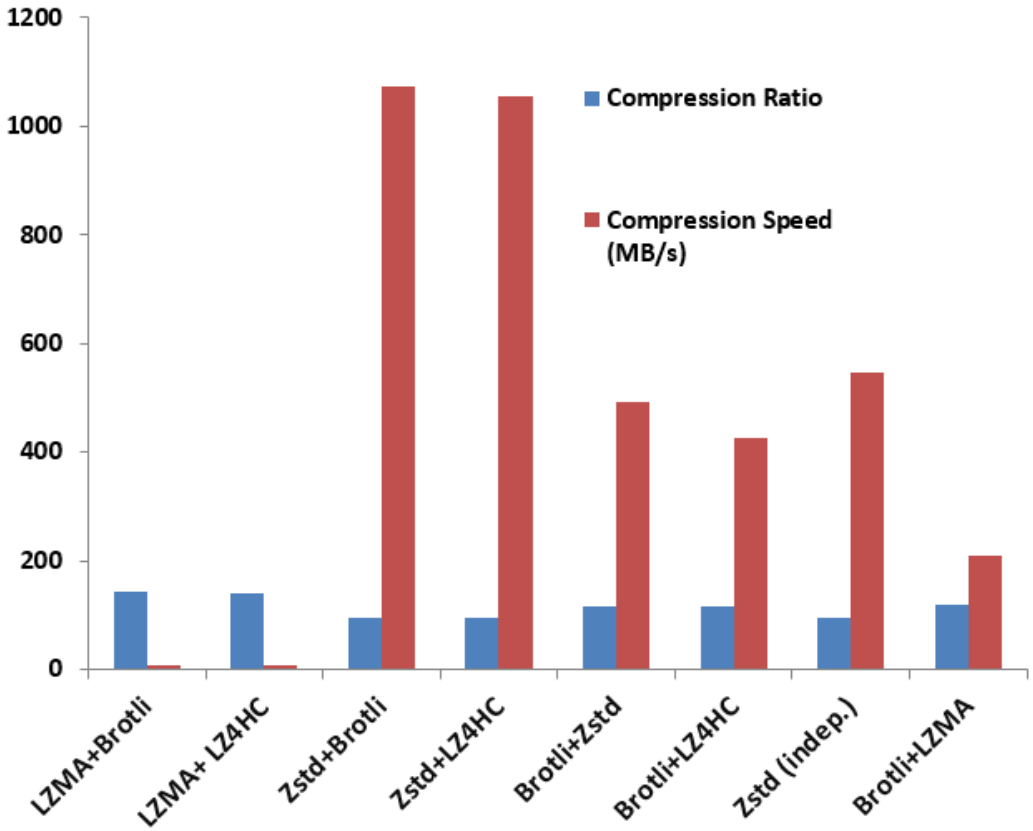

Fig. 7.2: Comparative analysis of Compression Ratio and Compression Speed

7.6.8 Frequency Analysis of Algorithm Participation

Looking at Fig. 7.3, we can observe that Zstd showed up the most in the top hybrid setups. It was used in almost half of the best pairs. Brotli came next, seen in about 37% of the top results, and Bzip2 followed close behind with nearly 33%. LZMA was not used much, showing up in only a few cases roughly 10%. This clearly tells us that Zstd plays a big role in hybrid compression. It blends well with other methods and keeps the results steady across all tests.

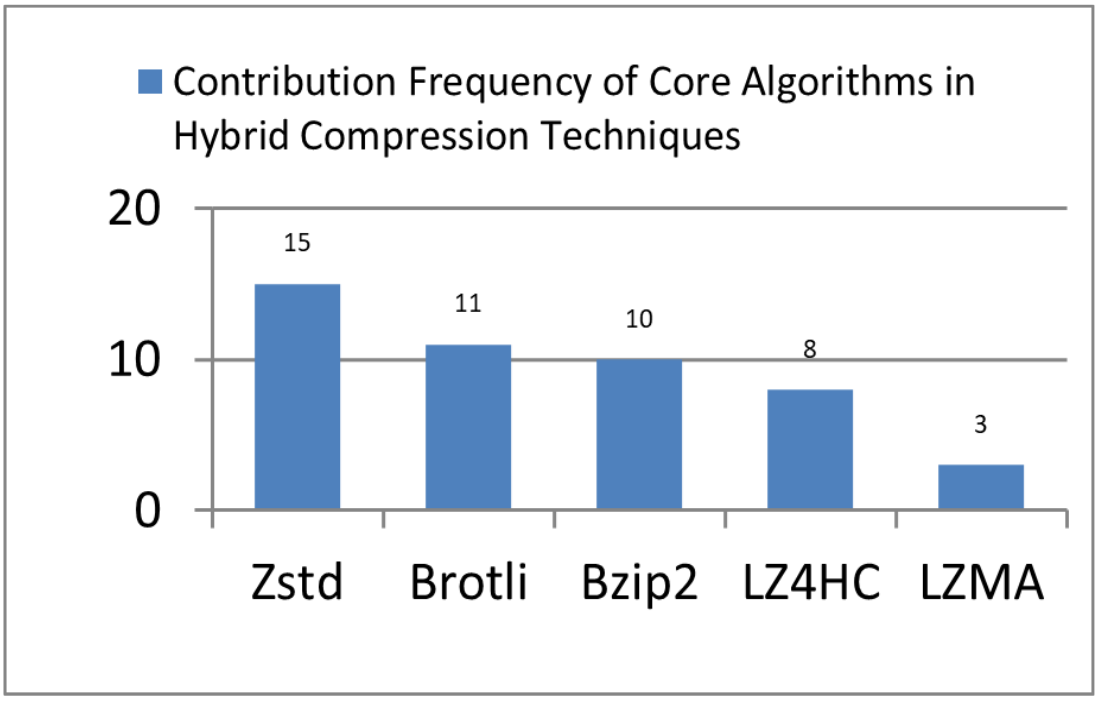

Fig. 7.3: Algorithm Participation in Top Hybrid Configurations

The Zstd+LZ4HC pair worked the best overall, giving strong compression, fast speed, and quick data recovery. The LZMA-based setups did give tighter compression, but they were much slower, so they fit better for long-term storage or backup work. On the other hand, Zstd-based hybrids ran faster and suited tasks that need quick responses.

Overall, hybrid compression seems to be a smart and workable choice for handling Hindi and other Indic-language data, especially on small systems where space, power, or speed are limited.

7.7 Limitations and Applicability

The large dataset in this chapter was synthetically expanded by repeating Hindi text segments, which amplified redundancy and resulted in unusually high compression ratios. These values should be seen as a scalability stress-test rather than a reflection of natural corpora. Baseline experiments on Kaggle Hindi datasets (Chapter 4) confirmed that the proposed methods retain their advantages under realistic conditions.

The present study has a few limitations that define the scope of its conclusions. The evaluation was restricted to Hindi text in the Devanagari script, and while this represents a major Indic language, the findings may not fully generalise to other scripts with different structural and linguistic features. The datasets considered were largely general-purpose, and the behaviour of the algorithms on highly specialised domains such as legal, biomedical, or social media text was not explored. Furthermore, the hybrid models were implemented as fixed two-stage cascades without adaptive selection, and the study was confined to lossless methods, leaving aside situations where carefully controlled lossy compression might be acceptable.

In spite of these boundaries, the work has strong applicability. LZMA-based hybrids, though slower, are well suited for archival storage where high compression is more valuable than speed. The proposed Zstd+LZ4HC hybrid is highly effective in large-scale and time-sensitive applications such as cloud storage, digital libraries, and streaming platforms, where both compression and decompression speeds are critical. Since the data can be decompress faster, this method becomes suitable for mobile apps, and for online learning platforms or digital dictionaries. It saves space and helps in language-related tasks, by reducing the text size and simplifying the file transfer. It can also be used in cloud or IoT setups, where both speed and energy saving are important.

7.8 Conclusion

During this phase, we discovered how Hindi text in Devanagari script can be compressed to save lot of storage space, with faster data transfer rate using hybrid lossless compression. We tried sixty hybrid setups on small, medium, and large data files. Among several of these tests, the hybrid compression demonstrated better results. They gave better and faster compression. Among all, the combination of Zstd and LZ4HC proved to provide the best results and gave consistent results across multiple executions.

Such hybrid models can easily be used in real world applications, like cloud storage, digital library systems, or any other tools designed for Indian languages. By adjusting the compression method for Devanagari text, this work clearly indicates towards better systems that support native languages in lucid and reliable way.

Taken together, the findings of this chapter validate hybridization as a viable pathway to more balanced and script-resilient compression. These observations provide a natural bridge to the next phase of this thesis, which moves from algorithmic cascades toward statistical techniques, specifically residual-based canonical Huffman compression, to further optimize memory and energy efficiency in constrained environments.

CHAPTER 8

Residual-Based Canonical Huffman Compression Techniques

8.1 Introduction

In the previous chapter, we examined hybrid pipelines that combine general-purpose compression algorithms to improve performance on Indic-script text. We now shift our focus to a domain-specific approach for time-series data, particularly sensor streams in Wireless Sensor Networks (WSNs), where energy, memory, and processing constraints impose unique challenges. Unlike text corpora, sensor data exhibits strong temporal correlation, which can be effectively exploited through residual transformation and compact entropy coding schemes.

Residual transformation step reduces the size of the data as it works by saving the difference between one sample and the next instead of using full values. So, most of the numbers stay close to zero, which makes it easier to compress. When followed by Canonical Huffman Coding, it builds a small and memory-saving design. Unlike Huffman coding this method does not need to store a full code tree, so it runs faster and uses very little space. That's really helpful for small sensor nodes that only have a few kilobytes of memory.

This chapter explains how residual encoding is joined with canonical Huffman coding. It is the base for the proposed Memory-Optimised Residual Adaptive Lossless Data Compression (MOR-ALDC) Algorithm. The method is made for WSNs, where sending data through radio uses the most power. It takes much more energy than local processing. So, we work on making the data smaller without losing any details, which helps save energy and makes the device last longer.

8.2 Research Objectives

This chapter matches the main goal of the thesis, which is to develop the efficient lossless compression method. The main objectives of this chapter are as follows:

1. To make a residual transformation and canonical Huffman-based design for lossless data compression that works well on low-resource WSN devices.

2. To develop a simple and memory-saving coding process that gives good compression without losing any information.

3. To test the proposed method with existing algorithms using both synthetic and real data, and to compare its CR, CS, memory and energy use.

8.3 Experimental Setup — Datasets & Metrics

To test the residual and canonical algorithm design, we used both synthetic and real-life sensor data. This helped to check how well it works in varying situations. The setup was made to test not only clean and controlled data but also noisy and uneven patterns that are common in WSNs.

8.3.1 Datasets

To test the MOR-ALDC, we used both synthetic and real datasets were used. This helped to study the performance in both controlled and real WSN conditions.

1. **Synthetic Dataset:**

A synthetic dataset was made to show different types of signals found in WSNs. It had four common patterns slow changes, sudden bursts, repeating signals, and random noise. These patterns are shown in Fig. 8.1.

a) Slowly Changing Signals:

A signal is said to be slowly changing if it shows small and slow changes in temperature, humidity, or light. They have a strong time link and are easy to predict.

b) Sudden Bursts:

A signal is said to sudden burst if it shows quick and big changes like machine start-up, fault, or human movement. They help test how fast the method reacts to sudden events.

c) Periodic Signals:

A signal is said to periodic if it repeats again and again over a certain period of time. like air conditioner cycles, tides, or machine operations. They help check how well the method handles repeating data over time.

d) Random Noise:

A signal is said to be random if there is uncertainty over the signal at any instant of time i.e., its instantaneous value cannot be predicted. These signals act like unwanted disturbances or sensor errors. They help test if the method can tell real data from noise.

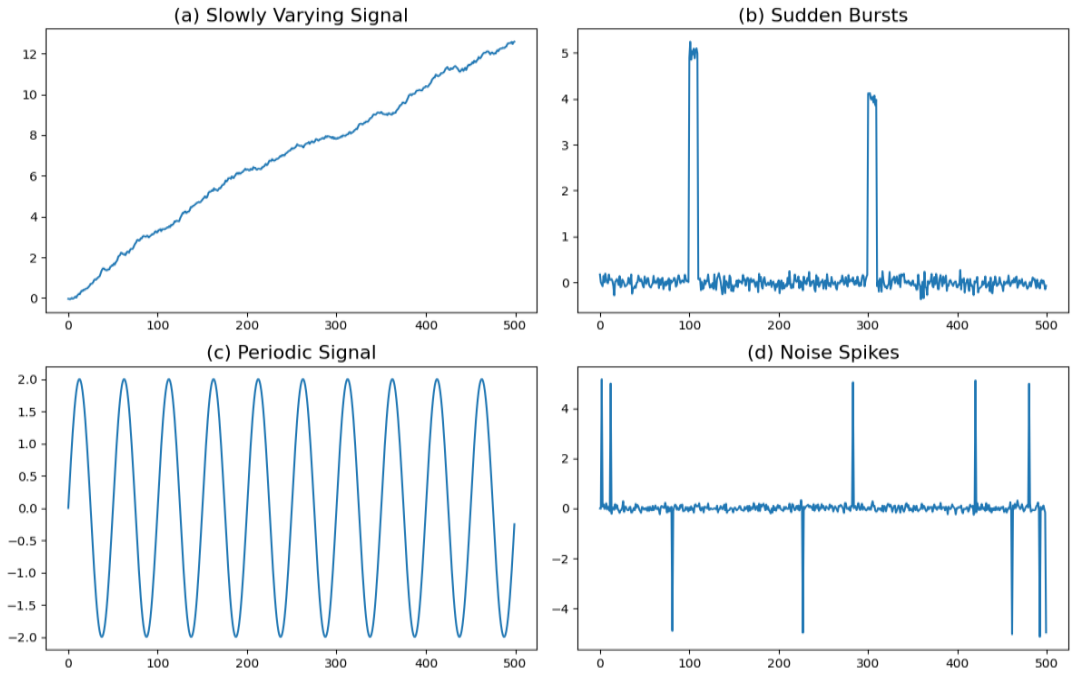

Fig 8.1: Synthetic WSN Signal Components used for evaluation: (a) Slowly Varying Signal, (b) Sudden Bursts, (c) Periodic Signal, and (d) Noise Spikes.

Each type of signal was made separately using simple NumPy functions such as sine, step, and Gaussian noise. The data was evenly taken and then changed into 16-bit values, to mimic the real life WSN sensors signals. After this, we joined all the signal parts together to form one complete time series to made a strong and repeatable dataset that could be used for testing.

2. Intel Lab Dataset:

The Intel Berkeley Research Lab dataset was used for real data testing. It contains long-term temperature readings collected from many sensor nodes as it has uneven changes and some noise, which makes it closer to real-life sensor conditions.

Both these datasets together make a balanced testing setup. The synthetic dataset is used for controlled and stress testing, while the Intel Lab dataset helps in checking real-world performance under noisy and mixed conditions.

8.3.2 Evaluation Metrics

The residual–canonical framework was evaluated using a set of standard and WSN-specific metrics to ensure fair comparison with conventional compression techniques.

i. **Compression Ratio (CR):** Defined in Eq. (3.1), CR reflects the efficiency of data reduction by comparing the size of the original input with its compressed output. A higher CR indicates greater effectiveness in minimizing payload size.

ii. **Energy Savings (ES%):** As given in Eq. (3.7), this metric quantifies the reduction in transmission energy achieved by compression, Where Eoriginal and Ecompressed denoting the energy required to transmit the original and compressed bitstreams, respectively. Since radio transmission is the most power-intensive operation in WSNs, this measure highlights the direct benefits of compression.

iii. **Encoding and Decoding Time:** Measured in seconds, this metric evaluates the computational feasibility of the framework when deployed on embedded sensor nodes.

8.3.3 Memory Metrics for WSN Implementation

As memory plays a key role in WSN nodes, two related measures were noted.

I. **Peak Working-Set RAM:** It is the highest amount of memory used by the compression code during its run. This value tells how much RAM a microcontroller needs to run the method properly. It was measured using Python's *tracemalloc* tool. It helps track the memory used by different parts of the algorithm such as residual buffers, Canonical Huffman tables, and bit-packing arrays.

II. **Resident-Set Size Change (RSS Δ):** The overall change in system-level memory usage, obtained via psutil. This includes additional pages mapped by the operating system or dependent libraries and therefore represents the worst-case memory growth during execution.

Together, these two measures provide a realistic view of both the internal memory footprint of the framework and its practical impact on the constrained memory budgets of WSN devices.

8.4 Method Overview

The proposed MOR-ALDC algorithm combines residual transformation with canonical Huffman coding. It forms a simple and lossless compression process made specially for WSN data. This method reduces repeated patterns over time and hence compresses the data using very little memory. It also helps save energy by sending smaller data packets. The full process is shown in Fig. 8.2.

8.4.1 Algorithm Overview

MOR-ALDC is a simple lossless compression method made for WSN sensor data. In such data, most readings are almost the same as the previous ones. So, the raw data is first changed into small differences, called residuals. After that, a light Canonical Huffman coding is applied to these values. This two-step method reduces the number of bits sent while adding only a little extra work for the node. It works in three simple steps:

1. **Residual Transformation**

Most environmental sensors change slowly over time. When we subtract each reading from the one before it, the result is mostly a small number close to zero. These small values are easier to compress than the original readings. Figure 8.2 shows this process using a sample input signal and its matching residual values.

Given a sequence of quantized readings

$$X=\{x_1, x_2 ,\ldots,x_n \}$$

were

X: complete input time-series of n sensor readings,

x_t: value of the sensor reading at discrete time step t, with t=1,2,…,n.

The residual signal r_t is computed using first-order differencing:

$$r(t) = x_t - x_{(t-1)} \quad \text{for t=2, 3..., n.} \qquad (8.3)$$

We set $r_1 = 0$

Here:

- r(t): residual (difference between consecutive samples).

This residual transformation exploits temporal redundancy in sensor data, yielding a signal where the majority of values are clustered around zero. This naturally reduces entropy and makes the data more amenable to compression using variable-length coding.

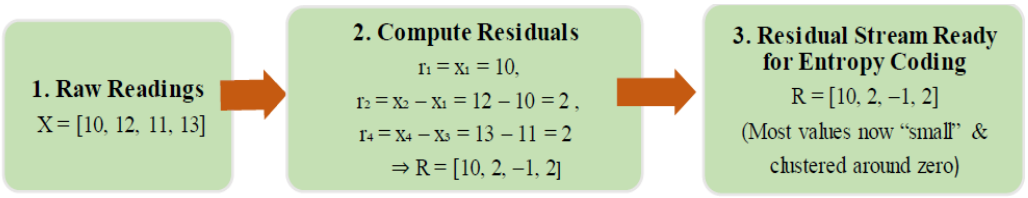

Fig. 8.2: Flowchart of the Residual Transformation Process in MOR-ALDC

2. Canonical Huffman Encoding

Once residuals are obtained, MOR-ALDC builds a frequency histogram and applies the two-queue Huffman algorithm to determine the optimal code-length for every symbol. These (symbol, length) pairs are sorted, and canonical Huffman codes are assigned shorter codes first, then symbols in lexicographic order. Because the decoder can regenerate every code from the length table alone, only this compact length table needs to be stored or sent. Each residual is then replaced by its canonical code and bit - packed into the output stream and gives the same compression result as the normal Huffman coding but needs only a few KB of memory. So, it can easily work on small WSN nodes with less memory. Fig 8.3 shows the four steps: (1) build the residual histogram; (2) derive the code-length table with the two-queue pass; (3) assign canonical codes; and (4) bit-pack the encoded residuals.

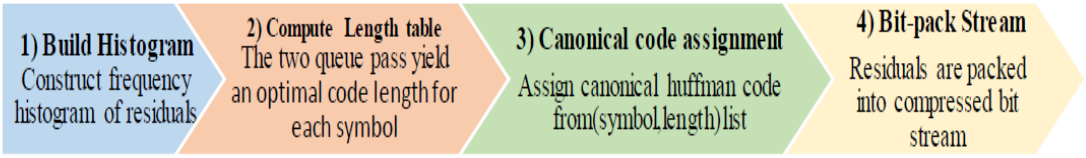

Fig. 8.3: Flowchart of the Canonical-Huffman encoding stage in MOR-ALDC

3. Transmission Energy Estimation

To measure how compression affects energy use, we used the first-order radio energy model. This model is commonly used in WSN research.

$$E_{tx} = b \times E_{elec} + b \times \varepsilon_{amp} \times d^2 \qquad (8.4)$$

Where:

E_{tx} : Total transmission energy,

ε_{amp} : Energy used by the transmitter amplifier per bit per m²,

E_{elec} : electronic energy consumed per bit (J/bit),

b : Number of transmitted bits,

d : Transmission distance (in meters)

This helps to compare the energy used before and after compression. Algorithm 8.1 shows the full steps used to implement the MOR-ALDC method as explained above.

Algorithm 8.1: Residual-Canonical Compression Pipeline

Algorithm 1: MOR-ALDC

Input: $X = \{x_1, x_2, \ldots, x_n\}$ // sequence of sensor readings
Output: C // encoded bitstream

1. Let R be an array of length n
2. R[0] ← 0
3. for i ← 1 to n−1 do
4. R[i] ← X[i] − X[i−1] // residual computation
5. end for

6. freq ← frequency table of all values in R
7. lengths ← TwoQueueHuffmanLengthPass(freq) // compute code lengths
8. mapping ← BuildCanonicalMapping(lengths) // assign canonical codes
9. C ← empty bitstream
10. for each r in R do

11. C. append (PackBits(mapping[r])) // bit-pack residual codes
12. end for
13. return C

8.5 Implementation and Experimental Protocol

The MOR-ALDC was implemented and tested along with other well-known lossless compression methods under the same conditions. This part explains the setup used for testing, how the benchmarks were done, and the steps followed to make the results fair and easy to repeat.

8.5.1 Experimental Configuration

All experiments in this chapter were done on a normal personal Laptop to keep the testing close to real research conditions. The system used a 13th Generation Intel® Core™ i7-1355U processor running at 1.70 GHz, with 10 cores and 12

threads. It had 16 GB of RAM, out of which 15.7 GB was usable. The computer used a 64-bit Windows 11 Home operating system (Version 24H2, Build 26100.4061) on a 64-bit x64 platform.

The programming work was done in Python 3.12. The main libraries used were NumPy, Pandas, Matplotlib, and the built-in versions of zlib, LZ4, and Zstandard for compression. All coding, debugging, and testing were done in PyCharm Professional 2025, which gave a stable platform for implementation. The datasets used for testing included the Intel Lab wireless sensor network data and the synthetic dataset discussed earlier in Section 8.3. This dataset had slow-changing, periodic, bursty, and noisy signals. We did not use CSV files. The sensor readings were scaled and saved as 16-bit values to keep the setup like real-world use as much as possible. Using this binary data helps remove extra file overhead and shows the real redundancy in the signals. Unless mentioned otherwise, all CR and results are based on these 16-bit binary streams. This makes sure that the testing of MOR-ALDC and other methods reflects real WSN conditions and not just file storage behavior.

8.5.2 Benchmarking Protocol

For comparison, the MOR-ALDC was tested along with three common lossless compression methods zlib, LZ4, and Zstandard (Zstd). All algorithms were made to work on the same sensor data so that the results stayed fair and balanced. The other codecs were used directly on the raw sensor data, while MOR-ALDC followed its two-step process of residual transformation and canonical Huffman coding. This approach was designed to make better use of the patterns found in WSN signals.

For fair comparison all tests were carried out under the same setup and system conditions. This helped to show the real difference in performance between the algorithms without any effect from data handling or system variation. The key features of these methods are given in Table 8.1.

Table 8.1: Characteristics of compression algorithms compared with MOR-ALDC.

Algorithm	Compression Type	Domain	Speed Profile	Preprocessing Used	WSN Suitability
Zlib	Lossless (DEFLATE)	General purpose	Moderate	None	Widely used; moderate compression and speed
LZ4	Lossless (LZ77-based)	General purpose	Very fast	None	Suitable for real-time WSN use due to speed
Zstd	Lossless (dictionary + entropy coding)	General purpose	Fast and balanced	None	Good balance of speed and compression
MOR-ALDC	Lossless (residual + canonical Huffman)	WSN-specific	Lightweight, linear	Residual transformation	High compression and energy savings, tailored for WSN workloads

8.6 Results and Discussion

To test the performance of the MOR-ALDC framework, experiments were done on both type of data synthetic and real. The results were compared with three commonly used codecs i.e. zlib, LZ4, and Zstd , under the same conditions. The comparison used four measures: CR, CS ,energy used for data transfer, encoding time, and memory use.

8.6.1 Synthetic Data Evaluation

The synthetic dataset was used to test the MOR-ALDC method under different signal conditions to check how the method performs under various data patterns found in WSNs

i. Compression Ratio Analysis

The results emphasised that MOR-ALDC gives a higher compression ratio than the other methods. As shown in Table 8.2, it reached 8.54×, which is better than Zstd (7.59×), zlib (7.42×), and LZ4 (4.9×). This better result comes from how the method groups similar residual values and uses Canonical Huffman coding efficiently. The improvement over LZ4 is large because LZ4 mainly focuses on speed and not on strong compression. Figure 8.4 shows the same trend by comparing the compression ratios of all the methods on the sample dataset.

Table 8.2: Compression and Energy Performance on Synthetic WSN Data

Algorithm	Compression Ratio (CR)	Original Energy (J)	Compressed Energy (J)	Energy Savings (%)
Zlib	7.42	0.0096	0.001294	86.52
LZ4	4.90	0.0096	0.001958	79.60
Zstd	7.59	0.0096	0.001265	86.83
MOR-ALDC	8.54	0.0096	0.001125	88.28

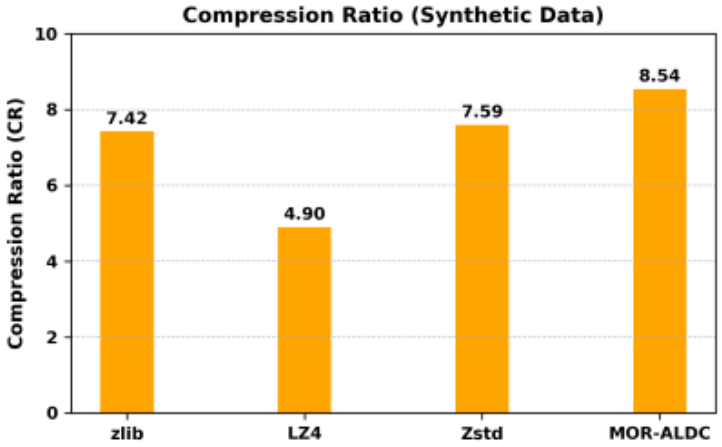

Fig 8.4: Compression ratio of algorithms on synthetic WSN data

ii. Energy Savings

The reduction in transmitted bits helped save a large amount of energy. As shown in table 8.2, MOR-ALDC gave the highest saving of 88.28%, which is slightly higher than Zstd, zlib, and LZ4. Figure 8.5 also shows this trend. Even small improvements in compression led to clear energy gains, which is important since sending data takes most of the power in WSNs.

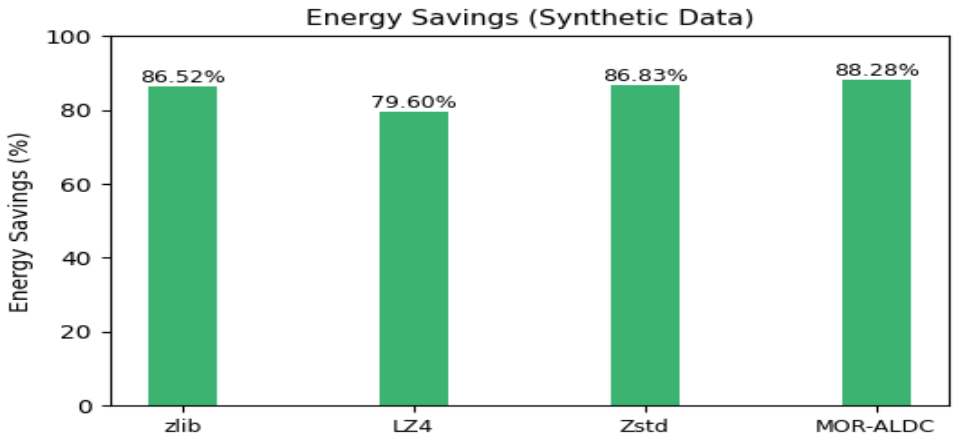

Fig. 8.5: Energy savings of compression algorithms on Synthetic WSN Data

iii. Runtime Performance

The other compression methods worked a little faster, taking around 0.0002–0.0003 seconds for 2000 samples. MOR-ALDC took about 0.0031 seconds, which means roughly 1.6 microseconds per sample. The comparison is summarized in Table 8.3.
This small delay is very minor compared to the normal working time of sensor nodes. So, the extra time is acceptable because it gives much better compression and saves more energy.

Table 8.3. Encoding Time Comparison (Synthetic Data, 2000 Samples)

Algorithm	MOR-ALDC	zlib	LZ4	Zstd
Encoding Time (s)	0.003148	0.000243	0.000300	0.000204

iv. Memory Usage

The memory test showed that MOR-ALDC used only about 3 KB of RAM at its highest point. The other methods needed much more, between 85 and 294 KB. Because of this small memory need, MOR-ALDC is more practical for small WSN nodes where memory is limited.

In short, the tests on synthetic data show that MOR-ALDC works better than the other common methods. It gives higher compression, saves more energy, and uses less memory, while adding only a very small delay in runtime.

8.6.2 Real-World Data Evaluation

To test how well MOR-ALDC works in real conditions, more experiments were done using the Intel Berkeley Research Lab dataset. This data was collected from 54 wireless sensor nodes over a long period. It includes readings of temperature, humidity, light, and voltage. For this study, the focus was mainly on temperature data.

(i) Residual Transformation

Figure 8.6 shows the effect of residual transformation on the real life data. The original signal changes slowly and has some noise, while the residual signal mostly stays close to zero with small variations. This shows that the readings are related over time, which helps in reducing data size.

Table 8.4 and Fig.8.7 shows the most common residual values, and more than 80% of them lie between −1 and +1. These results confirm that the data keeps strong time-based similarity, which makes it suitable for compression using Canonical Huffman coding.

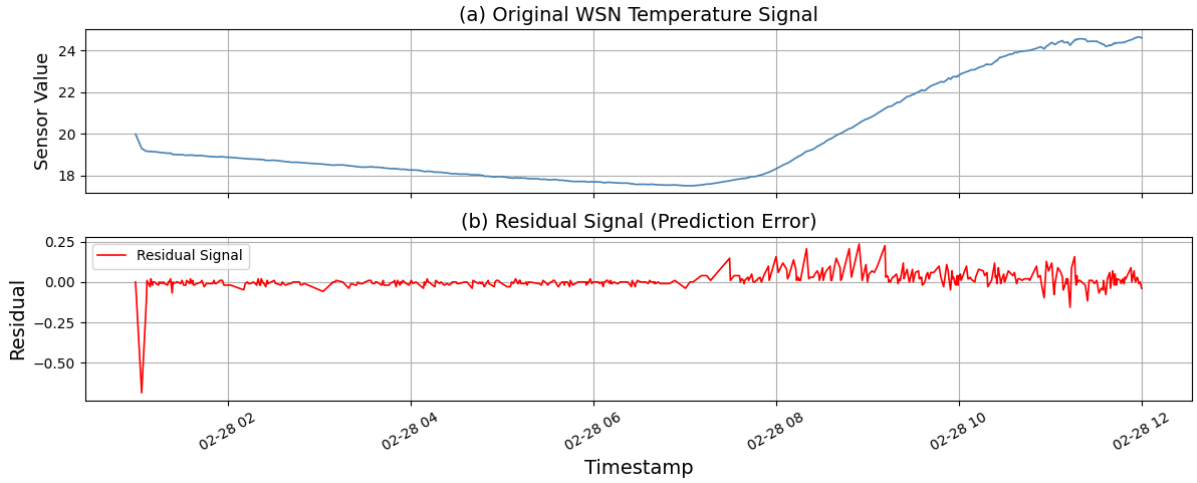

Fig. 8.6: Residual Transformation of Intel Lab WSN Temperature Data. (a) Original temperature signal, (b) residual signal after first-order differencing.

Table 8.4: Top 10 Most Frequent Residuals in the Intel Lab Real-World WSN Dataset

Residual Value	**0**	**1**	**−1**	**2**	**−2**	**−1004**	**3**	**1004**	**−3**	**1003**
Frequency	29342	2989	2985	360	321	142	137	137	115	112

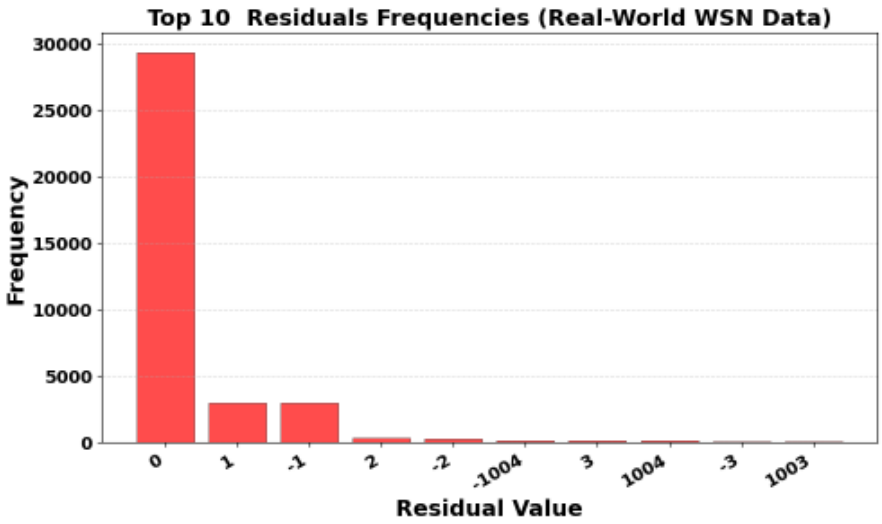

Fig. 8.7: Top 10 Residual Frequencies for Real-World Intel Lab WSN Data

(ii) Compression Ratio Analysis

The results on real sensor data showed a similar pattern to the synthetic tests. As seen in Table 8.5, MOR-ALDC reached a CR of 5.79×, which is higher than zlib (5.50×), Zstd (5.35×), and LZ4 (4.17×). Although the gap was slightly smaller, MOR-ALDC still gave better results. Figure 8.8 validate this trend and shows that its preprocessing step helps even when the data has noise or random changes.

Table 8.5 Compression Ratio and Energy Savings on Real-World WSN Data

Algorithm	Compression Ratio (CR)	Original Energy (J)	Compressed Energy (J)	Energy Savings (%)
Zlib	5.50	0.206626	0.037572	81.82
LZ4	4.17	0.206626	0.049567	76.01
Zstd	5.35	0.206626	0.038599	81.32
MOR-ALDC	5.79	0.206626	0.035686	82.73

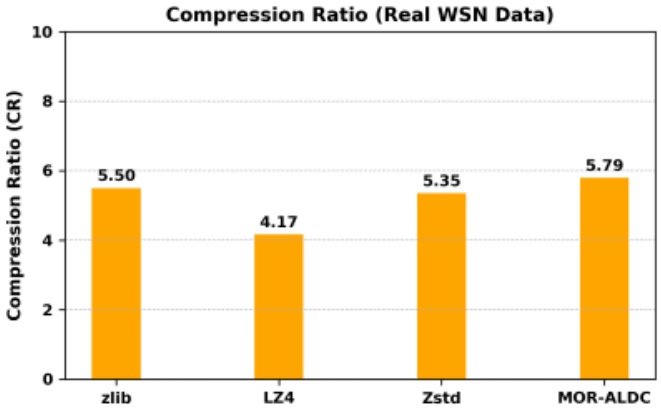

Fig. 8.8: Compression Ratio of Algorithms on Real WSN data

(iii) Energy Savings

The better compression also helped in saving more energy. As shown in Table 8.5, MOR-ALDC saved about 82.73% of transmission energy, which is slightly higher than zlib (81.82%) and Zstd (81.32%). LZ4 gave lower savings at 76.01%. Figure 8.9 also shows that MOR-ALDC can save more power in comparison to other while still keeping the data fully lossless.

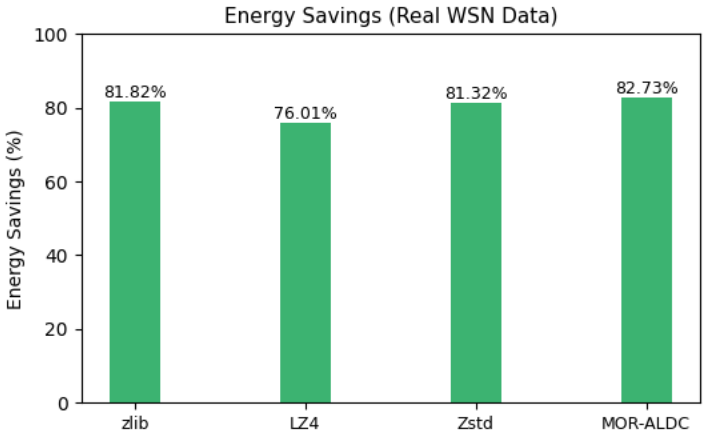

Fig. 8.9: Energy Savings of Compression Algorithms on Real WSN Data

(iv) Runtime Performance

Table 8.6 shows that the other compression methods worked faster on the Intel Lab data, taking about 0.004–0.006 seconds for 43,047 samples. MOR-ALDC took 0.065 seconds for the same data, which is around 1.5 microseconds per sample. Though it is a little slower, this extra time is very small compared to the sensor's working cycle. The time difference is acceptable because the method saves much more energy during data transmission.

TABLE 8.6: Encoding Time Comparison (Real-World WSN Data, 43,047 samples)

Algorithm	**MOR-ALDC**	**zlib**	**LZ4**	**Zstd**
Encoding Time (s)	0.065205	0.005986	0.006393	0.004090

(v) Memory Usage

The memory test showed that MOR-ALDC take very little space. It needed only around 16 KB of RAM at its highest use. As highlighted in Fig.8.10, MOR-ALAC in comparison with the other codecs used between 85 KB and 294 KB of memory. Because of its low memory need, MOR-ALDC can easily run on these small devices without disturbing how the sensors collect and share data.

8.6.3 Comparative Discussion

The overall results clearly show that MOR-ALDC gives better compression and energy savings than the other methods. Its design helps balance speed, memory, and power use, which is important for low-resource sensor nodes. In short, the method gives the best trade-off between performance and resource use, making it more practical for real WSN applications.

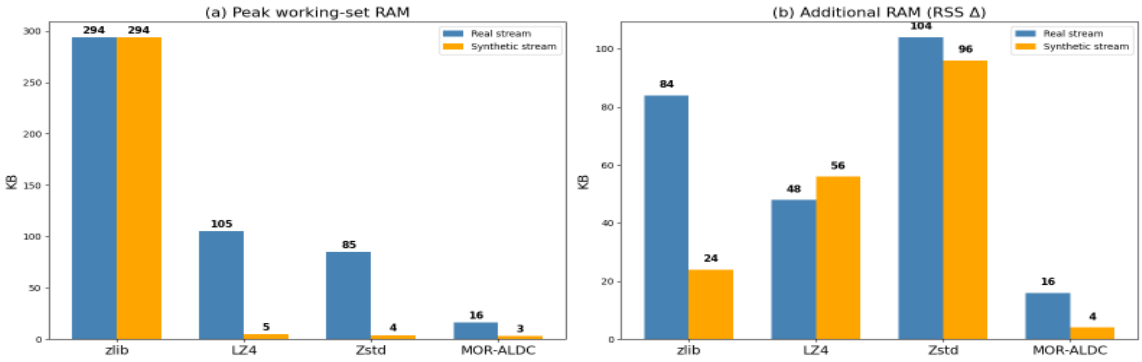

Fig. 8.10: Memory usage comparison across codecs for synthetic and real WSN streams: (a) Peak working-set RAM, (b) Additional RAM usage (RSS Δ)

Taken together, these results show that tailoring compression specifically to the properties of WSN signals brings significant benefits. MOR-ALDC achieves a practical balance of compression ratio, energy efficiency, and memory use, making it a strong candidate for deployment in real wireless sensor network environments. The run time of MOR-ALDC was a little longer than the fastest methods, but this small delay is acceptable because it gives much better energy savings.

8.7 Limitations and Applicability

The MOR-ALDC algorithm has two main limits. The one limitation is that it works best when sensor readings change slowly with time. When two readings are close to each other, the residual method gives good results. But if the readings are very random or have no pattern, the compression does not work as well.

The other limitation is that the system is rigid not flexible. Once the canonical Huffman codebook is made, it stays the same for all the data due to this the system cannot decide when to use compression or when to skip it. In the next chapter, we have improved it by adding AI-based learning so that it can adjust automatically.

Even with these limits, MOR-ALDC can be used in many real world applications. It works well for environmental data, smart farming, industry use, and health monitoring of buildings or machines especially where the data mostly changes slowly, with some sudden jumps or noise. For such type of data, the method gives good compression and saves energy as it runs easily on small WSN devices with less memory and power. So, MOR-ALDC is a good and practical choice for real-world use.

8.8 Conclusion

In this chapter we studied the MOR-ALDC algorithm in detail and showed how it works well for WSNs. The results proved that the method can be used easily in software and gives good performance. It meets its main goals high CR, low power use, and less memory need. In short, the software tests confirmed that MOR-ALDC is a suitable and efficient method for WSNs

The next chapter focuses on using hardware validation of MOR-ALDC on FPGA hardware. It explains how the simple design of this method can be used for real-time WSN. At this phase, we also make the system flexible by adding AIoT-based learning, so the system can decide on its own when and how to compress data. In this way, the benefits of MOR-ALDC can be used in more practical and changing IoT setups in the real-world conditions.

CHAPTER 9

Edge and FPGA Acceleration of AI-Guided Compression

9.1 Introduction

In the previous chapter, we showed that the MOR-ALDC algorithm runs quite well in software. Now the aim is to take it a step closer to real world use moving it towards edge and hardware levels. This stage of our work mainly looks at two sides: edge integration and FPGA acceleration.

On the edge side, we tried to bring the thinking part near the data source. Instead of sending every single reading to the cloud, the setup itself decides when to compress and by how much. It depends on a few things network speed, how the signal behaves, and how much energy the device still has. That way, bandwidth and power both get saved, and the key data remains ready when needed.

On the hardware front, we used an FPGA to make MOR-ALDC run live in real time. FPGAs can work in parallel, are flexible to modify, and need very little power perfect for small sensor nodes. We also put a few light ML modules next to the compression part, so the design can adjust on its own when the data or environment changes a bit.

This chapter is divided into two main parts. **Part A** explains our IoT-based edge framework for adaptive compression and security, developed and tested in a software setting. **Part B** describes the FPGA realization of MOR-ALDC with AI-guided modules, presenting the hardware side of feasibility. With these combined efforts, we have moved the framework a step closer to a working, deployment-ready system.

9.2 Research Objectives

The main objectives of this chapter are:

1. To designed an edge design for adaptive lossless compression of IoT data, allowing decisions near the data source to save energy, bandwidth, and time.

2. To ensured data integrity and security so that sensitive information remains accurate and safe during real-time transmission.

3. To implemented the MOR-ALDC core on FPGA hardware to validate that the method works in a compact, low-power, and real-time form for sensor networks.

4. To added AI-guided adaptiveness in the FPGA design to enable intelligent tasks such as signal classification, anomaly detection, and energy-aware operation.

9.3 Experimental Setup - Datasets & Metrics

We tested the framework in both edge and FPGA setups. The tests were done on a few different datasets using one common set of metrics. The idea was simple to see how the system behaves in conditions close to real IoT use. In real life, data doesn't come in one pattern; it changes a lot in type and speed. So, we picked datasets that had a mix of real-world and standard benchmark data. This helped us check how the setup reacts when the input changes. The same metrics were used for both software and hardware runs, just to keep the comparison fair and easy to follow.

9.3.1 Datasets

We used several datasets to test the framework in both edge and FPGA setups. For the IoT edge part, we worked on four kinds of data images, sensor readings, financial time-series, and text logs. These were picked because they show the kind of variety that usually comes in IoT systems. Some of it is multimedia, some is from sensors or structured records, and a few are just text-based logs. This mix gave us a fair idea of how the setup handles different types of information.

This way, we could see how the framework behaves under different data conditions, close to what happens in actual IoT networks.

For the AIoT–FPGA implementation, we have used the same datasets applied in earlier phases of this research. They include a synthetic dataset of 600 windows with 100 samples each, the Intel Lab dataset containing 34,443 temperature samples divided into 344 windows, and the UCI Air Quality dataset with 935,700 temperature readings split into 93 windows. Each signal was processed in fixed-length windows of 100 samples to maintain uniform evaluation and fair comparison across all methods. The preprocessing steps and dataset details have already been discussed in Chapter 3.

9.3.2 Evaluation Metrics

We have evaluated the IoT edge framework using three main measures. The first was the CR, which shows how much data reduction was achieved. The second focused on energy savings, calculated as the percentage reduction in energy use for different data types. The third measure checked the accuracy of reconstructed data using Root Mean Squared Error (RMSE) and Peak Signal-to-Noise Ratio (PSNR).

For the AIoT–FPGA setup, we looked at both software and hardware sides together. In our tests, we checked the compression ratio (CR), time taken per window, and energy used (in mJ) to see if the design really suits low-power IoT devices. We also kept track of the memory use and other hardware things like resource count, delay, and power draw. These FPGA readings came from Vivado runs on the Artix-7 board, which gave a good idea of how the design might behave in real conditions.

9.4 Part A: IoT Edge Framework

9.4.1 Method Overview

We made the IoT edge framework to make data transfer more energy-efficient and secure. It's built with three main layers — the IoT Device Layer, the Edge Layer, and the Cloud Layer. Each one plays its own part so that compression stays useful, dependable, and safe for real-world systems.

1. IoT Device Layer

Here, the raw data comes directly from IoT sensors and devices. The packets are taken in real time without affecting how the device normally works. In our setup, this part was made inside a controlled software setup. We used Wireshark to collect the packets and then got them ready for the next step of processing.

2. Edge Layer

This is the main working part of the framework. It includes three small modules that handle compression, data quality, and security together:

a. Adaptive Compression: This module changes the compression strength based on the network speed and how capable the device is. It helps keep a good balance between saving energy and keeping the processing light.

b. Quality Adjustment Control: In this part, we used a few small machine learning checks to keep track of how accurate the compressed data stays. The module tweaks the compression settings on the go, making sure that the key data stays clear and correct when it matters.

c. Security Protocol: This part keeps the data safe and private. It uses encryption, anonymization, and digital signatures to protect the packets from possible cyber threats.

All these modules were made and tested using Python. This setup helped us simulate the full edge-processing workflow in a realistic and easy to control software environment.

3. Cloud Layer:

In final step, we have sent the processed packets to the **cloud layer**. This layer handles long-term storage and performs higher-level analytics. It helps generate real-time insights and supports automated decisions without adding any delay to device operations.

The complete workflow of the framework is shown in **Fig. 9.1**. It presents how data moves from collection at IoT devices, through adaptive compression and quality control at the edge, and finally reaches the cloud for secure transmission and analysis.

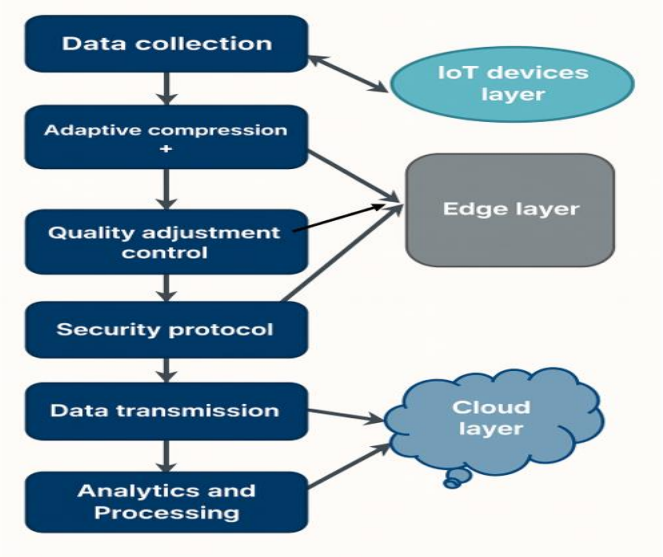

Fig. 9.1: Layout Of the Proposed Iot Based Edge Computing Framework

- **Illustrated Example**

We have explained the working of the framework through a simple example. Imagine a sensor that records temperature in a room. At the device layer (see Fig. 9.1), these readings are grouped into data packets. Once the packets reach the edge layer, three steps take place:

1. Adaptive compression reduces the packet size. When the network is busy, the system compresses more; when it is free, compression is lighter. The process remains lossless, so the original temperature values can always be recovered.

2. Quality control verifies that the compressed data matches the original readings exactly.

3. Security protection is applied to keep the data safe during transfer.

After these steps, the cloud layer receives the compressed and secure data for storage or further analysis. This setup keeps the data small and safe while staying accurate. At the same time, it cuts down a bit on energy and bandwidth use.

9.4.2 Implementation and Experimental Protocol (IoT Edge Framework)

We implemented the IoT edge framework in a controlled software setup to see how well it can actually work under real-world conditions. At the device level, raw IoT traffic was created and captured with Wireshark. The packets we collected, shown in Fig. 9.2, were then used as input for the edge-side processing.

At the edge, the data passed through three small Python modules that handled adaptive compression, quality checks, and security. This kind of setup made the testing easier to repeat and simpler to verify later when we move it to hardware.

The datasets and metrics used here were the same ones explained earlier in Sections 9.3.1 and 9.3.2. Keeping them the same helped maintain consistency and avoided any change in the evaluation process.

For benchmarking, we compared our method with some well-known techniques like ADCA, RCQA, and SCTP. We also looked at the more recent works by Nasif, Chang, Tosoni, and Qiu to make the comparison fair and complete.

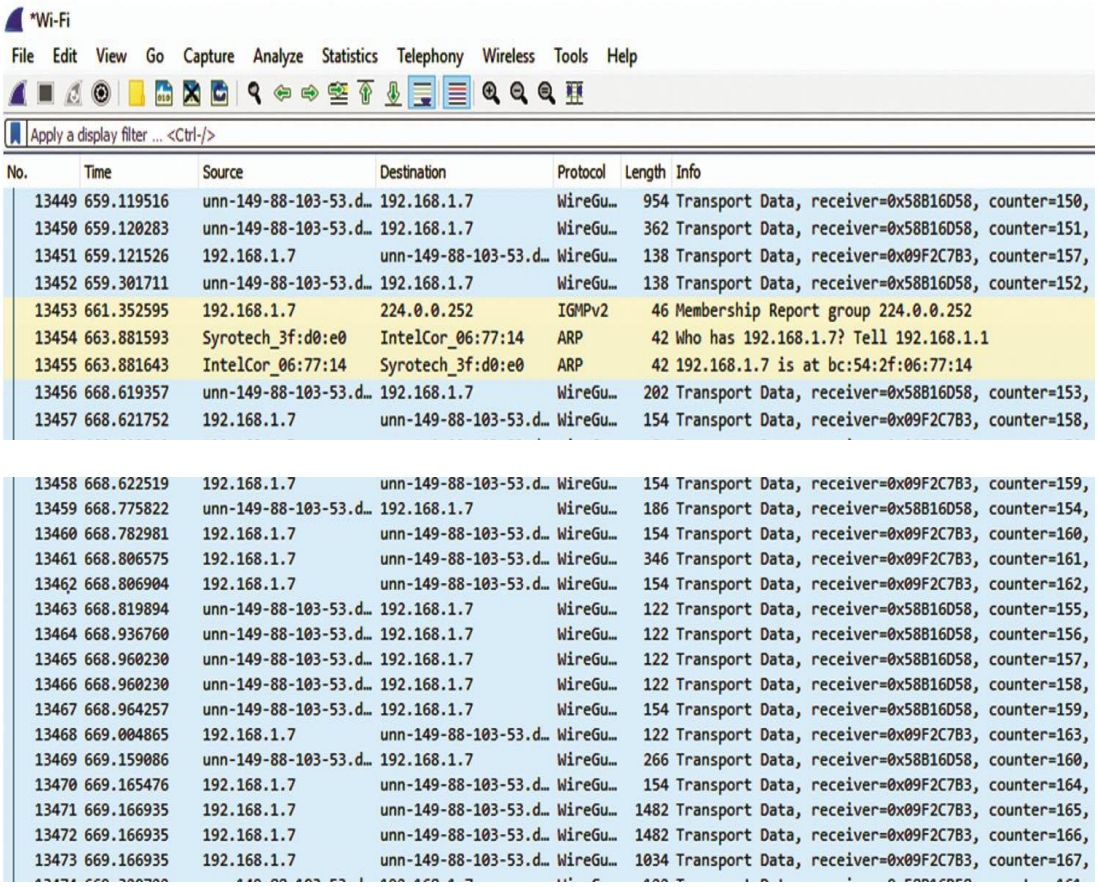

Fig. 9.2: Iot Sensor Packets Captured at the Device Layer using Wireshark

9.4.3 Results and Discussion of IoT Edge Framework

When tested under the same setup, the proposed IoT edge framework was compared with both recent studies and standard algorithms. The comparison covered works by Nasif et al. (2024), Chang and Sobelman (2024), Tosoni et al. (2025), and Qiu et al. (2025), along with three known baselines — ADCA, RCQA, and SCTP. We used the reported values from these studies directly so that the evaluation stays fair and open.

Under identical testing, our framework gave the best results overall. It reached an RMSE of about 0.6653 and a PSNR of 92.14, which clearly shows better reconstruction accuracy. In other methods, RMSE was mostly between 1.25 and 1.85, and PSNR stayed below 90. The lower RMSE here means less difference from the real data, and the higher PSNR shows that the reconstructed signals stayed quite clear.

Table 9.1 gives a summary of these results, and the full layered setup can be seen in Fig. 9.1. Apart from accuracy, the framework also saved energy, which makes it a good fit for edge-based IoT systems. From these tests, it looks clear that the design works well under different setups and can be used in real IoT conditions.

Table 9.1: Comparative analysis of the proposed approach with existing compression algorithms and recent studies

Approaches / Algorithms / Recent Studies	**Avg. RMSE**	**Avg. PSNR**
Nasif et al. (2024)	1.8562	80.02
Chang et al. (2024)	1.7433	86.43
Tosoni et al. (2025)	1.5341	84.45
Qiu et al. (2025)	1.4873	88.09
ADCA	1.4567	79.12
RCQA	1.3612	83.76
SCTP	1.2583	82.35
Proposed Lossless Compression	**0.6653**	**92.14**

The energy results in Table 9.2 clearly show the strength of the proposed method. For image data, the power use was roughly 92% without any compression, about 89% with moderate compression, and dropped to nearly 80% with the proposed setup. A similar pattern showed up for sensor, financial, and text data too. On average, the saving was close to 10–15% over moderate compression and around 20% when compared to sending the data raw.

Table 9.2: Power consumption analysis of the proposed approach based on compression level

Data Type	**No Compression**	**Moderate Compression**	**Lossless Compression (Proposed)**
Image data	92%	89%	80%
Sensor data	95%	91%	85%
Financial data	98%	95%	90%
Text data	94%	92%	88%

These results confirm that the setup delivers higher reconstruction accuracy and substantial energy savings compared to conventional methods. It seems to fit nicely in IoT setups where we need both good accuracy and careful use of power and memory

9.4.4 Limitations and Applicability

There are a few limits noticed in this study. Lossless compression works well only when the data has some kind of pattern or repetition. When the signal is too random or has high entropy, compression doesn't help much because there isn't enough redundancy to remove. Another issue comes with choosing the right parameters, especially for image data. Keeping good quality while trying to get high compression needs a bit of careful tuning. If that balance goes off, the output might not meet the needs of sensitive or critical applications.

Still, the proposed IoT-based lossless compression setup has many possible uses. In general systems, it can work like ZIP compression to store or send data safely and efficiently. In fields like healthcare, it can help compress and encode patient records or medical images, where accuracy can't be compromised. Apart from such single uses, the same approach also helps the wider IoT setup by cutting down the raw data sent to the cloud. This, in turn, reduces the pressure on data centres and saves energy, which makes large-scale IoT systems more sustainable in the long run.

9.5 Part B: AIoT–FPGA Framework

9.5.1 Method Overview

In this part, we took the earlier MOR-ALDC framework and made it smarter by adding some intelligence at the edge. Its compression core was also made faster using FPGA. The new version, called AIoT–MOR-ALDC, was built in three main steps: codec testing, AI-based adaptiveness, and finally the hardware setup.

Step 1: Codec Benchmarking and Backbone Selection

In the first step, we tested a few different codecs after applying residual transformation to the signals. The idea was to see which one could work best as the main backbone for the system. After several trials, Canonical Huffman coding turned out to be the most balanced option. It gave a good mix of compression, speed, and low memory use, making it a natural choice for the setup. These features make it a good fit for small and resource-limited IoT nodes.

Step 2: AI-Guided Adaptiveness

In this stage adaptiveness was added with the help of three small and simple

machine learning modules. Each signal window, having 100 samples, was processed one after another in sequence.

- A Random Forest classifier identified the pattern of the signal, such as smooth, burst, drift, or noisy.
- An Isolation Forest anomaly detector, using an eight-dimensional feature vector, marked any unusual behaviour to ensure that critical information was retained.
- A Logistic Regression model then combined the feature set and anomaly scores to decide whether compression would save energy.

When compression gave a clear benefit, the data window was encoded using MOR-ALDC. If not, it was sent directly in its raw form. To check how well the system performed, the AIoT–MOR-ALDC approach was compared with four commonly used codecs Huffman, RLE, FELACS, and Rice.

- **Illustrated Example**

We took a simple temperature-monitoring example. A sensor records 100 readings, which together make one data window. The pattern classifier identifies this window as *periodic* when the values show a smooth daily change, or *noisy* when they fluctuate irregularly. Next, the anomaly detector checks for unusual activity. For instance, if a heater is turned on and the temperature spikes suddenly, the event is marked as an anomaly. After that, the energy-aware model decides whether compression is worthwhile. When the network is congested, compression helps save both bandwidth and power, and the data is encoded using MOR-ALDC. If the signal is already simple or compression offers little benefit, the raw data is sent directly. Through this process, we have ensured that normal data is compressed, anomalies are preserved exactly, and unnecessary computation is avoided when compression provides no advantage.

Step 3: FPGA Realization

In the last stage, the MOR-ALDC compression core was implemented on an FPGA platform. The design was written in VHDL and synthesized on a Xilinx Zynq-7020 board using the Vivado toolchain. The hardware works in real time, it calculates residuals, aligns them using registers, and maps them to Canonical Huffman codes stored inside a small ROM.

To make the setup ready for real-time IoT applications, the pipeline was tuned to keep a steady two-cycle encoding delay. This hardware test shows that the MOR-ALDC algorithm can run efficiently on reconfigurable hardware, even when resources are limited.

9.5.2 Implementation & Experimental Protocol of AIoT–FPGA Framework

The three sequential stages of implementation of the AIoT-MOR-ALDC approach are

i. Codec Benchmarking and Backbone Selection

In the first stage we focused on **benchmarking multiple codecs** to find a good backbone for the system. Before adding **AI-guided adaptiveness**, a detailed study was done on different compression methods after applying **residual transformation** to the input signals. Five codecs were tested under the same setup **Canonical Huffman (MOR-ALDC backbone)**, **RLE + Huffman**, **Rice coding**, **Zlib**, and **LZ4**.

The general-purpose algorithms like **Zlib** and **LZ4** gave strong compression but needed more memory and power, which makes them less suitable for small **IoT nodes**. On the other side, simple methods like **RLE** and **Rice coding** used less energy but showed weaker compression. The **Canonical Huffman** method gave the right balance reliable compression, fast runtime, and low memory use. It is again proved that the **Residual Transformation + Canonical Huffman** combination was selected as the backbone codec for the **AI-guided compression pipeline** and **FPGA setup.**

ii. Codec Selection for AIoT Evaluation

In this stage, the AIoT–MOR-ALDC setup was tested against four common codecs Huffman, RLE, FELACS, and Rice. These cover different ways of doing compression in IoT and WSN systems.

To keep things fair, all codecs ran with their default settings on the same datasets. That made the comparison honest and easy to judge. The evaluation was done in two parts, first, to pick the best backbone codec, and then, to check how the proposed AIoT-MOR-ALDC performs against the rest. In the end, the framework came out strong. It worked smoothly, showed reliable results, and proved more adaptive and efficient than the others.

iii. AI Module Training and Evaluation

Three small and simple machine learning models were trained and tested to guide the adaptive behaviour of the system. Each signal stream was split into separate, non-overlapping windows of 100 samples, which acted as the basic unit for both training and testing. The models worked one after another in sequence, as shown in Fig. 9.3**.**

- **Pattern Classifier (Random Forest)**

A Random Forest classifier was trained on raw 100-sample windows using an 80/20 train–test split. The model had 100 trees with depth settings tuned through cross-validation. It reached 100% accuracy, correctly identifying six signal types smooth, periodic, drift, burst, noisy, and mixed.

An example of one of the decision trees from this classifier is given in **Appendix II (Fig. I.1)**. It shows how different **WSN signal windows** were divided into their respective categories.

(b) Anomaly Detector (Isolation Forest)

For anomaly detection, each data window was converted into an **8-dimensional feature vector** that captured both statistical and dynamic properties, as described in **Table 9.3**. The Isolation Forest model, using **100 estimators** and a **contamination rate of 0.05**, successfully separated normal windows (**95%**) from anomalous ones (**5%**).

(c) Energy Decision Model (Logistic Regression)

In the last stage, a logistic regression model was trained on the same feature vectors, combined with the anomaly scores**.** L2 regularization was used in the logistic regression model to keep the weights small and make the energy-saving predictions more stable and reliable. it predicted whether compression would help save energy. On the test data, the model suggested compression for **98.8%** of the windows and skipping for **1.2%**, showing a clear preference toward energy-efficient operation.

The training results and performance values for all three AI modules are shown in Table 9.3. The models gave stable and accurate results across all tests, which shows that they are reliable and suitable for use in the AIoT–MOR-ALDC setup.

Table 9.3: Training Details and Performance of AI Modules

Module	Input Features	Training Details	Performance (Test Set)
Pattern Classifier	Raw 100-point signal windows	Random Forest, 80/20 train-test split, 100 trees, depth tuned via 5-fold CV	100% accuracy; perfect separation of 6 classes
Anomaly Detector	8-D statistical feature vector	Isolation Forest, 100 estimators, contamination = 0.05	Normal: 570 (95%), Anomalous: 30 (5%)
Energy Decision Model	8-D feature vector + anomaly score	Logistic Regression, L2 regularization (C=1.0), 5-fold CV	Compress: 593 windows (98.8%), Skip: 7 (1.2%)

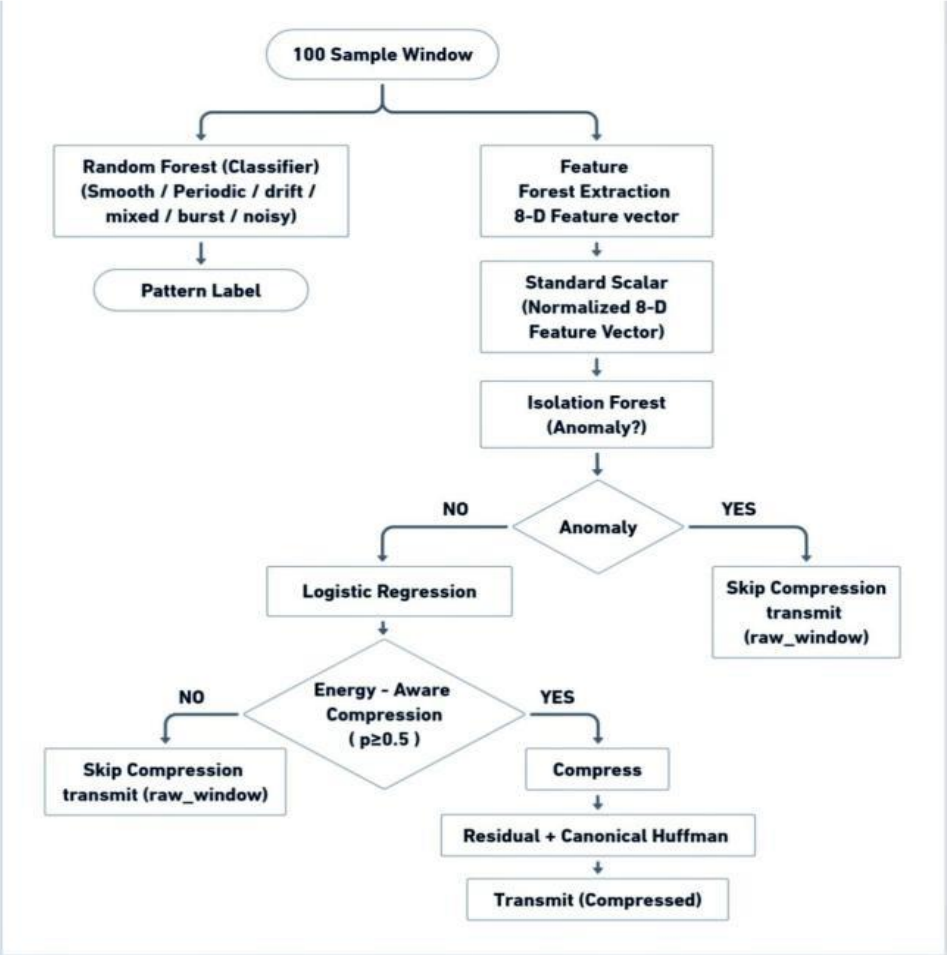

Fig. 9.3: Flowchart of AI-guided Adaptiveness in the AIoT–MOR-ALDC Framework

- **Statistical Features for Anomaly Detection**

Each signal window of 100 samples was represented by an 8-dimensional feature vector that captured both statistical and dynamic characteristics of the signal. These features, listed in Table 9.4, provide a compact summary of the data and help the anomaly detector distinguish normal behaviour from abnormal activity.

Table 9.4: 8-D Statistical Features Extracted from each Signal Window

Feature	**Description**	**Purpose in anomaly detection**
Mean	Average of all samples in the window	Captures the overall signal level
Standard Deviation	Measure of variation around the mean	Detects fluctuations and instability
Skewness	Degree of asymmetry in the data distribution	Identifies bias towards high or low values
Kurtosis	Measure of "peakedness" or flatness of distribution	Highlights sharp peaks or heavy tails
Peak-to-Peak Range	Difference between maximum and minimum values	Flags extreme deviations within the window
Average Slope	Mean rate of change between consecutive samples	Reflects overall trend or drift in the signal
Zero-Crossings	Number of times the signal crosses zero	Indicates oscillatory or periodic behaviour
Total Energy	Sum of squared values of the samples	Represents overall signal strength and bursts

i. Software and FPGA Environment

We have carried out the experiments in the same standard setup described earlier in Chapter 3 (Methodology). The FPGA synthesis was performed using Xilinx Vivado 2023.2, where the MOR-ALDC compression core was written in VHDL and mapped onto a Xilinx Zynq-7020 device. The design was first verified through RTL simulation to confirm correct functionality. After that, synthesis was completed to ensure proper timing and resource utilization. The AI modules continued to run in Python, following the same environment used in the earlier stages of this study.

ii. Experimental Protocol

All experiments were done in the same standard setup mentioned in Chapter 3. For the software part, the AI modules and codec wrappers were written in Python and run on a workstation under fixed settings.

For the hardware part, the synthesized design was loaded onto the FPGA and tested using the synthetic, Intel Lab, and Air Quality datasets. The main performance checks included CR, runtime, energy use, and memory footprint. These tests helped keep the comparison across codecs and datasets fair and consistent. The idea was to make sure that the results stayed clear and not affected by any bias.

9.5.3 Results and Discussion of AIoT–FPGA Framework

The AIoT–MOR-ALDC framework was tested in both software and hardware setups to check how well it works in real use. The results cover different parts codec benchmarking, AI-guided adaptiveness, dataset-level performance, and FPGA realization. Each part was studied carefully, and the main findings were discussed together with the observed outcomes.

1. Pre-AI Codec Benchmarking

Before adding the AI-based adaptiveness, five codecs were tested after applying residual transformation Canonical Huffman (MOR-ALDC)**,** RLE + Huffman**,** Rice**,** Zlib**,** and LZ4**.** As seen in Table 9.5**,** the general-purpose codecs like Zlib and LZ4 gave better compression ratios but needed more time and memory**.** On the other hand, lighter schemes such as RLE and Rice worked faster but showed poor compression results.

Out of all, the Canonical Huffman (MOR-ALDC) method gave the most balanced outcome, combining good compression with low runtime and minimal memory use. Because of this balance, it was selected as the main backbone codec for the AIoT–MOR-ALDC setup, especially for low-power WSN systems.

Table 9.5: Pre-AI codec benchmarking results on synthetic dataset (residual-transformed signals)

Codec	Avg CR	Avg Time (ms)	Avg Energy (mJ)	Avg Memory (KB)
MOR-ALDC (Res+CH)	**5.29×**	0.3501	0.01751	0.21
RLE + Huffman	4.02×	0.2733	0.01367	0.31
Rice	2.06×	0.1150	0.00575	1.92
Zlib	2.33×	0.0217	0.00109	0.19
LZ4	1.79×	0.0099	0.00050	0.38

2. Dataset-Level Performance of AIoT–MOR-ALDC

a) Synthetic Data

After adding AI-guided adaptiveness, steady performance gains were seen across all datasets, as shown in Table 9.5. On the synthetic WSN dataset, the AIoT–MOR-ALDC setup showed clear improvement over the baseline codecs. As shown in Table 9.6, it reached an average compression ratio of 10.02×, which is more than double that of Huffman (4.76×) and much higher than Rice (2.91×). Though RLE and FELACS worked faster, their compression stayed very low (0.5×), which limits their use in real IoT setups.

The framework used only 0.0063 mJ per window, showing strong energy efficiency compared to its compression gain. Its memory use stayed low at 0.81 KB, much smaller than Rice (6.08 KB) and close to Huffman. The runtime of 6.35 ms was a bit higher, but still fine considering the better compression and lower data transfer load.

Overall, the AI modules added no real overhead. These results show that intelligent, model-based control improves compression, saves energy, and keeps the design sustainable for IoT use.

b) Intel Lab Dataset:

The results for the Intel Lab dataset are shown in Table 9.7. Here, the AIoT–MOR-ALDC setup did really well, reaching a compression ratio of 20.37×. That's way higher than Huffman (7.71×), Rice (2.99×), and the light ones like RLE or

FELACS, which stayed close to 2×. So, roughly a three times improvement over the best of the older methods.

Table 9.6: AIoT–MOR-ALDC versus standard codecs on synthetic WSN dataset

Codec	Avg CR	Avg Time (ms)	Avg Energy (mJ)	Avg Memory (KB)
Huffman	4.76×	0.28	0.00028	0.41
RLE	0.50×	0.02	0.00002	0.35
FELACS	0.50×	0.04	0.00004	0.03
Rice	2.91×	0.14	0.00014	6.08
AIoT–MOR-ALDC	**10.02×**	6.35	0.00635	0.81

The runtime came out a little higher, about 4.71 ms, but the memory use stayed very low at around 0.8 KB. Energy use was also tiny — just 0.000707 mJ. All these numbers are easily within what small IoT devices can manage, and the trade-off feels fair considering how much data it saves during transmission.

From these tests, it's clear that for real-world temperature readings, the AIoT–MOR-ALDC adapts nicely to how the data behaves. It keeps important changes, cuts the redundant parts, and still runs efficiently. Overall, the setup looks practical and ready for real IoT edge use.

Table 9.7: AIoT–MOR-ALDC versus standard codecs on Intel Lab dataset

Codec	Avg CR	Avg Time (ms)	Avg Energy (mJ)	Avg Memory (KB)
Huffman	7.71×	0.13	0.000019	0.80
RLE	2.11×	0.02	0.000003	0.06
FELACS	2.00×	0.02	0.000003	0.09
Rice	2.99×	0.12	0.000019	3.90
AIoT–MOR-ALDC	**20.37×**	4.71	0.000707	0.67

c) Air Quality Dataset:

The results for the Air Quality dataset are listed in Table 9.8. This one is quite noisy and changes a lot, but even then, the AIoT–MOR-ALDC setup worked nicely. It reached an average compression ratio of 6.77×, which is better than Huffman (5.10×) and Rice (3.84×). The improvement here is not as big as what we saw with the Synthetic or Intel Lab data, but it still shows that the system can deal with messy, real-world signals where noise makes compression tough.

The framework ran with a runtime of 4.38 ms and used only 0.000657 mJ of energy both are well within what normal IoT boards can handle. The memory need was around 1.93 KB, a bit more than on simpler datasets but still okay for small edge devices. RLE and FELACS, on the other hand, barely did any compression (roughly 0.54×), which again shows how this approach keeps a good balance between efficiency and accuracy.

Even under such noisy data, the adaptive modules helped it hold steady performance without putting much load on the system. It stayed reliable not only in test runs but also in conditions closer to real IoT work, like air quality or weather monitoring, where readings keep changing every few seconds.

Table 9.8: AIoT–MOR-ALDC versus standard codecs on Air Quality dataset

Codec	Avg CR	Avg Time (ms)	Avg Energy (mJ)	Avg Memory (KB)
Huffman	5.10×	0.20	0.000030	0.13
RLE	0.54×	0.06	0.000008	0.00
FELACS	0.54×	0.02	0.000003	0.22
Rice	3.84×	0.12	0.000018	4.90
AIoT–MOR-ALDC	**6.77×**	4.38	0.000657	1.93

These results establish that adaptiveness is not only useful for synthetic datasets but also generalizes well to diverse and noisy real-world workloads.

3. Impact of Adaptiveness

The impact of adaptiveness in the AIoT–MOR-ALDC system can be noticed clearly in Figs. 9.4 and 9.5. These results show how the setup decides on its own when to apply compression and when to just skip it. This kind of smart selection helps the system run more efficiently overall.

As seen in Fig. 9.4, the number of windows chosen for compression was different for each dataset. In the Synthetic dataset, about 90.7% of the windows got compressed, which makes sense since the data there is regular and quite predictable. But in the Intel Lab and Air Quality datasets, the numbers dropped to around 63%. Real data is usually noisy and irregular, so compression doesn't always give much gain.

Still, even with fewer windows being compressed, the total compression ratio actually went up ` that's visible in Fig. 9.5. For example, in the Intel Lab dataset, it jumped from 20.37× (all windows) to 31.42× (only the compressed ones). For Air Quality, it rose from 6.77× to 10.09×, while the Synthetic data saw a small bump from 10.02× to 10.60×. These results make it clear that the adaptive logic focuses effort only where it counts, saving time and energy on the rest.

This kind of smart selection is quite useful for IoT edge devices**.** By avoiding low-yield segments, the system saves both energy and processing time. It also helps extend the battery life of sensor nodes while still keeping the data accurate and reliable.

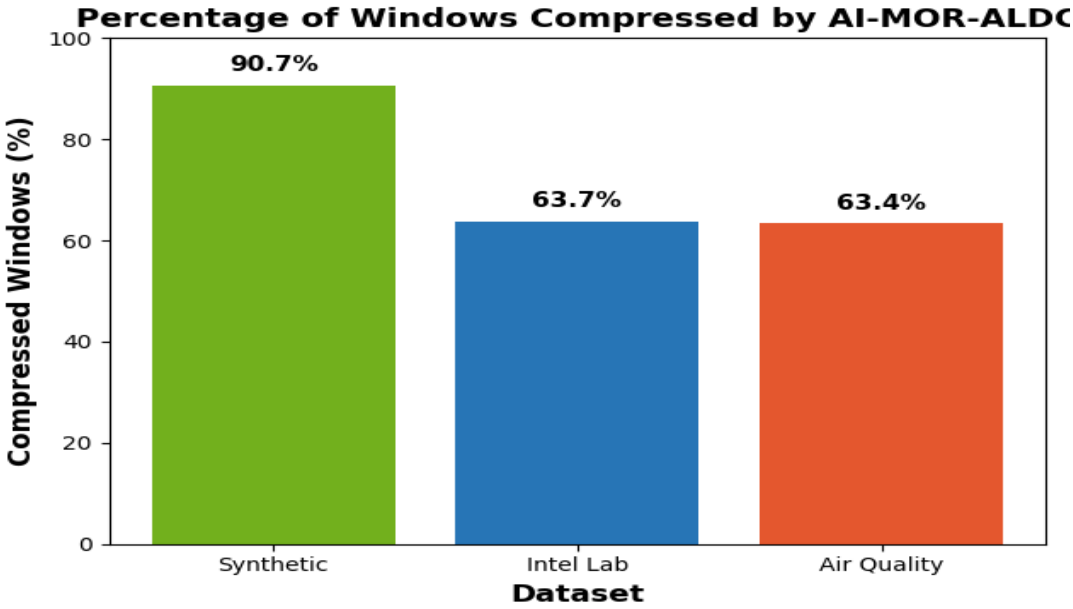

Fig. 9.4: Percentage of Windows Compressed by AI–MOR-ALDC across Synthetic, Intel Lab, and Air Quality datasets.

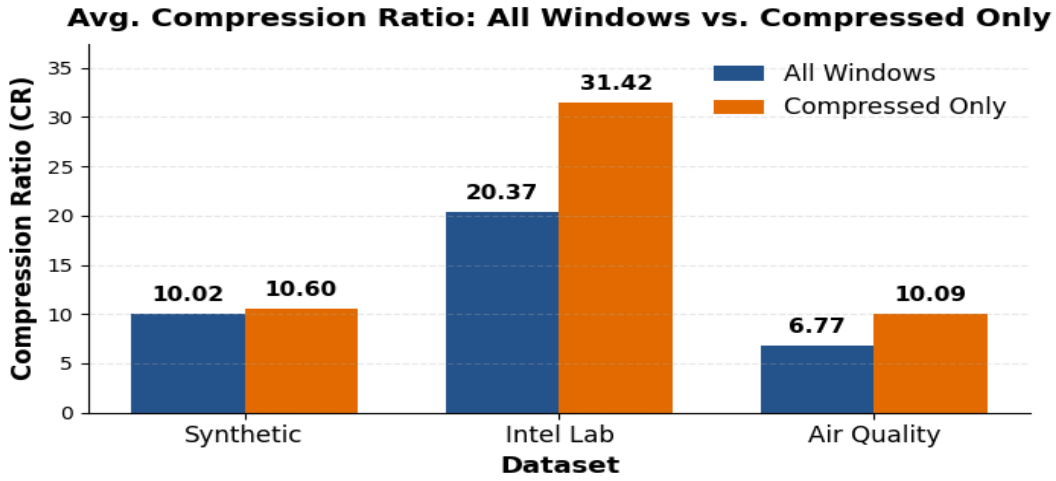

Fig. 9.5: Effect of Adaptive Selection on average Compression Ratio Achieved by AIoT–MOR-ALDC across Datasets

4. FPGA Realization

To check the hardware feasibility of the design, the MOR-ALDC core was synthesized on a Xilinx Zynq-7020 (Artix-7) FPGA using the Vivado toolchain. The RTL schematic of the design is shown in Fig. 9.6**.** The process starts by storing the previous input sample**,** then calculating the residual**,** and finally mapping it to canonical Huffman codes kept in a small ROM**.**

This setup shows that the complete compression process can be done through **a** compact and efficient hardware pipeline**.** The design fits well for IoT and WSN platforms where both power and memory are limited.

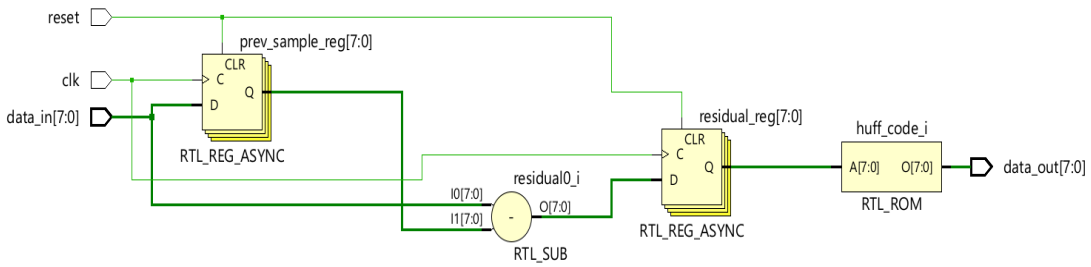

Fig. 9.6: RTL Block Diagram of the Compressor

The functionality of the design was checked using RTL simulation. As seen in **Fig. 9.7**, the compressor gives one encoded symbol every two clock cycles. The timing waveforms confirmed that the **residuals** were computed correctly and the Huffman codes were fetched accurately. The latency stayed stable and predictable throughout the test. These results show that the compression core works well in **real-time streaming**, where a steady output rate really matters.

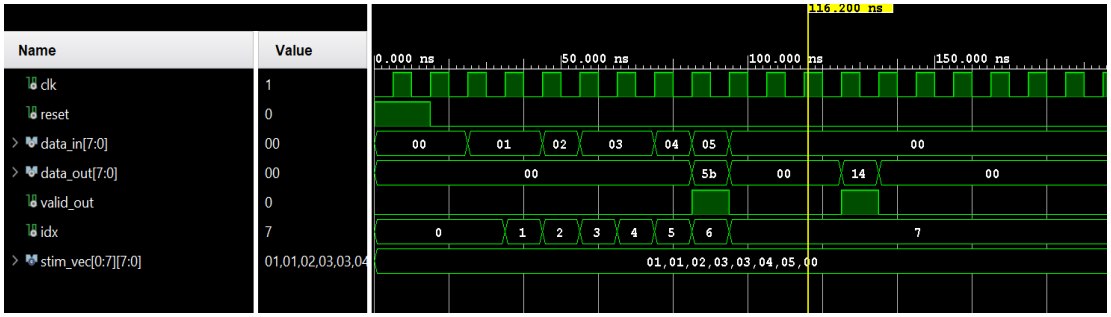

Fig. 9.7: Functional Simulation Waveform Showing Two-Cycle Latency and Correct Code Lookup

After the simulation, the design was synthesized to check its hardware footprint. The post-synthesis utilization report, given in Table 9.9 and shown in Fig. 9.8, shows that the setup uses very few resources only 41 LUTs (0.51%), 37 flip-flops (0.23%), and 19 I/Os (12.67%). It is important to note that the design does not use any BRAM or DSP blocks, which means it is lightweight and fits well within the limited resources of typical IoT nodes.

Table 9.9: FPGA Resource Utilization of MOR-ALDC Core (Post-Synthesis, Xilinx Zynq-7020)

Resource	Utilization	Available	Utilization (%)
LUTs	41	8000	0.51 %
FFs	37	16000	0.23 %
IO	19	150	12.67 %

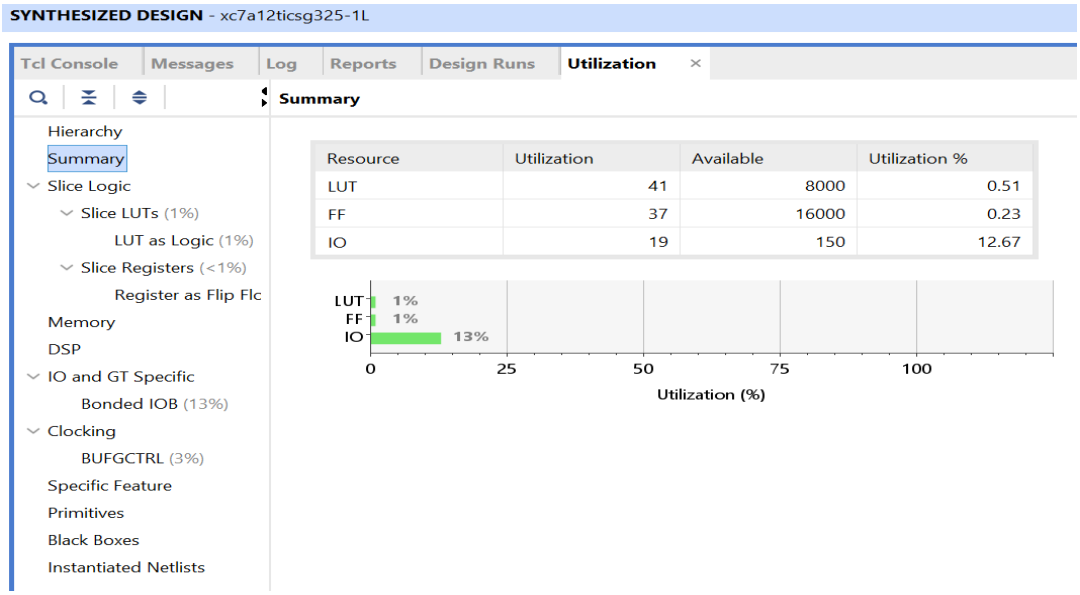

Fig. 9.8: Post-synthesis FPGA resource utilization of the MOR-ALDC core on Zynq-7020.

These results show that the MOR-ALDC core can be implemented on FPGA easily and with very low hardware cost. By moving the compression task to hardware and keeping the AI modules in software, the setup achieves a good balance between hardware and software. It keeps real-time performance steady while still allowing the flexibility needed for AI-based control. This makes it practical for modern IoT systems that need both speed and adaptability.

Taken together, a few main outcomes stand out. First, the AI-guided adaptiveness helps improve efficiency by compressing only the data windows that give high returns. This selective method boosts the compression ratio and saves energy by avoiding useless computation. Second, the MOR-ALDC codec performs better than both general-purpose and WSN-based methods. It gives a good mix of compression gain, runtime, and memory use across all datasets. Lastly, the FPGA results confirm that the system is ready for actual use. It needs very few hardware resources, runs with stable latency, and works well for low-power IoT devices in real time.

In short, these findings show that the AIoT–MOR-ALDC framework performs effectively in software and remains lightweight in hardware. We have demonstrated that it successfully bridges the gap between simulation and deployment, providing a practical and adaptive solution for IoT data compression.

2.11.1 Limitations and Applicability

i. Limitations

Most of the limitations identified for the software implementation of MOR-ALDC in Chapter 8, such as reduced effectiveness on highly noisy or weakly correlated signals, remain applicable here and are not repeated in detail. At the AIoT level, two additional considerations arise. First, the inclusion of lightweight AI modules (pattern classifier, anomaly detector, and energy model) introduces a modest runtime and energy overhead. While this overhead is small relative to the compression gains, it may be significant in ultra-constrained devices. Second, the current hardware–software partition places the AI modules in software while mapping only the compression core to FPGA. This setup keeps the design flexible, but it may not work well for very small sensor nodes that have limited processing power.

ii. Applicability

Even with a few limits, the AIoT–MOR-ALDC framework stays quite useful in areas where both efficiency and flexibility matter. It works nicely for environmental monitoring like tracking temperature, humidity, or air quality. The same setup can also be used in smart farming, industrial IoT, or predictive maintenance, for example in vibration or energy monitoring. It also suits structural health systems where data needs to be compressed but the key changes or faults must still be kept. The FPGA part further proves that the system can actually run in real time at the edge. It shows that this setup can help IoT devices last longer on battery and respond faster, even when power and bandwidth are very limited.

9.6 Conclusion

This chapter is the last stage of the work, where the MOR-ALDC system finally moved from plain simulation to something that can actually be deployed. The IoT edge setup showed that adaptive and secure lossless compression can run right at the data source. It helped save energy and bandwidth while still keeping the data quality almost the same.

After that, the AIoT–MOR-ALDC version added a bit of intelligence using machine learning and also got a speed boost through FPGA. It gave strong compression results and used very little hardware, which makes it fit well for real IoT systems.

Overall, the results show that the whole approach is not just solid in theory but also practical to use. It looks like a good step forward for upcoming IoT and WSN applications.

The results from this part complete the testing of MOR-ALDC at all three levels software, edge, and hardware. With its performance now well proven, the next chapter brings everything together and talks about the main contributions of this work. It also looks at what can be done next to take this research a step ahead.

CHAPTER 10

CONCLUSION, FUTURE SCOPE, AND SOCIAL IMPACT

10.1 Conclusions

This chapter brings together the main findings of the work done .In this thesis, we studied efficient lossless data compression techniques and developed a new algorithm for IoT-based WSNs .Throughout this research, the work moved in phases from reviewing existing techniques in modern day life especially for WSNs, gaps where identified ,benchmarking was done , the testing of hybrid methods, and then to design new algorithms and its hardware validation in VHDL using FPGA.

In phase 1 we worked on improvement in algorithm efficiency of LZW. The LZWP version showed that even small changes in the dictionary updating methods can increase the processing speed significantly without reducing the compression ratio. The MOR-ALDC model incorporated the joining of residual transformation with canonical Huffman coding. It helped to lower memory use and consume less power ideal for working in constrained environment. From these results we find that with smart design, complex algorithms can be customised to meet specific requirements.

Next, we moved to the semantic based approach. As we all know data does not always follow one form, so we explored best possible way to match the nature of different data types. The ontology-driven methods, such as OntoRLE, showed that adding light semantic structure improves compression without changing the meaning of the data. In the next phase, hybrid cascaded methods were used on Devanagari-based Hindi text. Results show that combining two or more algorithms together often works better than a single algorithm.

The most important part of this thesis is the focus on practical testing. Compression is valuable only if it can be used in real life conditions, under real constraints. Software based AI-guided decision module and hardware-based compression on FPGA using VHDL shows that compression can be both adaptive and energy-aware. This structure decides when compression needs to done, and how best to apply it under varying bandwidth and power conditions which give it a upper edge from the static based methods. The successful hardware testing of the prototype confirmed that these ideas are not limited to the lab, but can be added to edge–cloud systems to save work and help devices run for a longer time.

Overall, the thesis improves the efficiency of algorithms, adapts compression to the demands of different conditions, and brings these methods closer to practical validation. The work also shows that efficient compression techniques can make a difference in society. The fulfillment of the research objectives defined in Chapter 1 is summarised in Table 10.1, which maps each objective to the corresponding phase and contribution of the thesis.

Table 10.1 Mapping of Research Objectives to Phases and Contributions

S.No.	Objective	Fulfillment (Phase & Contribution)
1.	Preprocessing for dataset performance	Phase 3 → Ontology-driven healthcare & OntoRLE for WSN
2.	Throughput-oriented architecture	Phase 2 → LZWP (Time-optimized LZW)
3.	Efficient memory organization for IoT/WSN	Phase 5 → MOR-ALDC (Residual + Canonical Huffman)
4.	Compression for next-gen systems	Phase 4 → Hybrid pipelines (Zstd→LZ4HC, Indic scripts, IoT)
5.	Energy-efficient FPGA techniques	Phase 6 → AI-guided FPGA + Edge framework

Through these outcomes, the research successfully meets its stated objectives and contributes to the advancement of efficient lossless compression techniques that are both theoretically sound and practically deployable.

10.1.1 Limitations

While the results reported in this thesis are encouraging, several factors limit the generalisability of the claims. The experiments carried out on synthetic, Intel-Lab and air-quality data, were largely confined to controlled or small-scale deployments; multi-month, large-scale field trials are needed to characterise model drift, maintenance overheads and sustained energy savings in the wild. The AI-guided decision module depends on representative, labelled data for initial training; its behaviour under scarce labels or shifting data distributions calls for lightweight, bounded-memory online adaptation strategies. Hardware validation was limited to FPGA prototypes, so translating the designs to ASICs, estimating production costs and assessing long-term industrial reliability remain open engineering tasks. Finally, although privacy and security concerns are discussed in the thesis, integrating compression-friendly authentication/encryption schemes and performing formal threat modelling were outside the present scope.

10.2 Future Work

While this thesis has advanced the study and development of efficient lossless data compression, research in this area remains an evolving journey. Each stage of the work opened up questions and possibilities that could not be fully addressed within the present scope. Building on the contributions made, several promising directions can be pursued in the future:

1. **On-node continual learning for compression decisioning**

The AI-guided decision module developed in Chapter 9 improved the selection of algorithms, but it remained static once trained. A natural extension is to design bounded-memory, online update mechanisms that allow nodes to adapt their compression choices as data patterns evolve in the field. This would make compression decisions more resilient to distributional drift in real deployments.

2. **Dynamic AI-guided hybrid pipeline orchestration**

The current decision module focused on choosing a single algorithm based on bandwidth and energy conditions. Future work could extend this idea to a lightweight runtime that dynamically selects or composes multi-stage codec cascades (e.g., Zstd→LZ4HC, MOR-ALDC→Huffman). Such orchestration would jointly optimise compression ratio, throughput, and energy efficiency, allowing each node to handle heterogeneous workloads more effectively.

3. **Semi-automatic ontology induction for broader applicability**

The ontology-driven methods in Chapter 6 improved compression but required manual effort in designing ontologies. Future research could explore semi-automatic tools that suggest ontology classes and relations from data patterns, with experts refining the results. This assisted approach would reduce the human burden and make semantic compression practical across diverse application domains.

4. **Cross-sensor and multimodal compression**

Thus far, the focus was on single data streams. Extending residual transforms and coding schemes to jointly compress correlated streams—such as temperature, humidity, and vibration—offers an important next step. By exploiting inter-sensor correlations, cross-sensor or multimodal compression could further improve efficiency and reduce redundancy in real deployments.

5. **Hardware refinement toward ASIC and ultra-low-power RTL**

The FPGA prototypes built in Chapter 9 demonstrated feasibility, but ASIC or highly optimised RTL designs would deliver even lower power and cost for mass deployment. Future efforts could explore hardware–software co-design strategies such as mixed-precision arithmetic, clock gating, and near-memory computation to achieve ultra-low-power performance.

6. **Joint compression–security designs for sensitive domains**

In healthcare and industrial IoT, confidentiality is as important as efficiency. A promising direction is to integrate compression with lightweight security mechanisms, such as authenticated compressed formats, selective masking, or compress-friendly

encryption. These designs would protect sensitive data without sacrificing compressibility.

7. Long-term field trials and operational studies

Real world validation is very important after the prototypes trial. Multi-month pilot deployments whether in campus networks, agricultural monitoring, or healthcare settings should be carried out to test the full pipeline (OntoRLE, MOR-ALDC, and AI decisioning). Such tests would show how model changes with time, upkeep issues, and real energy savings in the teal life conditions.

10.3 Contributions of the Thesis

1. Benchmarking classical algorithms

A set of tests was done on English text, sensor data, and Hindi text to measure standard methods such as LZW, Zstd, Deflate, and BZIP2. Results showed that while they work well on normal computers, they slow down and use more power on embedded systems. This gave a clear reason to design lighter and adaptive models for IoT and WSN setups.

2. Development of LZWP

A new method called LZWP was made to make compression faster without losing quality. It uses a better prefix setup and word-level coding. It worked faster than the old LZW method and was good for tasks that need both speed and trust.

3. Ontology-based semantic methods

Two simple meaning-based methods were created. The first, **OntoRLE**, joins basic ontology rules with run-length encoding for sensor data. The second works on healthcare records, keeping the main meaning while removing repeated parts. These show that adding meaning helps make files smaller while keeping them easy to read.

4. Hybrid compression pipelines

Several mixed pipelines such as Zstd + LZ4HC and BZiP2 + Brotli were tested on

Hindi and IoT datasets. They reduced delay and processing load but still gave good compression. Using more than one stage proved better than one-algorithm setups, especially for real-time and low-power devices.

5. MOR-ALDC framework

The **MOR-ALDC** algorithm follows residual transformation then canonical Huffman coding. It gives high CR less power use and optimized memory when tests on synthetic and real-world temperature sensor data. This makes it a good and practical choice for small devices.

6. AI-guided FPGA system

An AI-based FPGA setup was made to test the practical viability of the algorithm on real hardware. The AI part runs in software and chooses the compression settings. The FPGA part takes care of the data work. This hardware-software partitioning design enables practical, adaptive, and energy-aware compression suitable for real-world WSN applications. This setup showed that adaptive compression can run well on edge devices without any drop in performance.

An AI-based FPGA setup was built to test the ideas in hardware. The AI unit runs on software and decides the compression settings, while the FPGA handles the data part. It proved that adaptive compression can work on edge devices without losing performance.

10.4 Social Impact

When lossless compression becomes faster and easier to use, it changes how people share and use information. The methods made here save energy, reduce cost and directly or indirectly protect the environment. They can also work with many types of data. Because of this, the study can support fields such as healthcare, environment, education, and communication.

1. **Environmental Monitoring and Sustainability**

The methods in this work, such as OntoRLE and MOR-ALDC, due to its low power need fit well for long-term use. These systems often run in far or sensitive places where it is not easy to replace batteries or fix hardware. By lowering transmission needs and saving power, the methods here can make the systems last longer and protect the environment. It may also lead to integration of several IoT devices for minimizing load on data centres thereby enhancing energy consumption in networks.

2. **Affordable and Reliable Healthcare Solutions**

It will also be used to encode patients' record in healthcare systems which require compression and encoding of images which leads to hassle free treatment of the patients even in rural areas also. It also helps in developing tailored and patient centric equipment. It may also lead to integration of various medical devices for enhancing patient outcomes and providing better access to healthcare. It can also be utilized for application in patient health monitoring systems in order to ease the complex treatment procedure.

3. **Support for Indic Languages and Digital Inclusion**

The hybrid cascaded compression techniques can lower storage and transmission costs a critical factor for digital libraries, government archives, mobile platforms, and cloud-based services that operate under constrained budgets and infrastructure. As a result, it can support projects in education, government work, and public communication.

Fast data and small files help people use regional language content, even with slow internet. This helps more people learn and share information.

4. **Smarter and Greener IoT Ecosystems**

The AI and FPGA systems in the thesis works on adaptive data compression design. It does not have to follow one fixed rule. The AI based decision module can decide when and how much to compress data in real time, depending on power and network use. This in return can save energy and other resources. It also helps reduce costs and lower pollution from digital systems and make them more reliable.

In different fields like disaster work, factory safety, transport, and defence, reliability matters as much as speed. The lossless compression methods made in this study help improve both speed and stability when resources are limited. It is used in ZIP file

format to compress data and make it available to users in an efficient and secured manner In real use, this can lead to faster warning systems, safer work, and stronger setups that protect people and support the country.

10.5 Concluding Remarks

This study shows that efficient lossless compression can be upgrade to meet the needs of changing needs of the world. It can be adapted to work fast WSNs. It is also flexible to fit many kinds of use and strong enough for real life applications. The aims set at the beginning have been met, giving results that are useful and practical. As digital data keeps growing day by day, the ideas in this work can guide future research and new designs. They also help in handling the immense amount of data more stable, open, and sustainable.

REFERENCES:

1. Abishek Sudarshan. *Healthcare Analytics Dataset*. Kaggle. Available from: https://www.kaggle.com/datasets/abisheksudarshan/healthcare-analytics.accessed on (23 April 2023).

2. Ahmad, M.S., Lata, S., Mehfuz, S. and Ahmad, A., 2019. Lossless Compression Algorithm for Energy Efficient Wireless Sensor Network, *2019 International Conference on Power Electronics, Control and Automation (ICPECA), New Delhi, India,* pp. 1–4, https:/ /DOI.Org/10.11 09/IC PECA47973.2019.8975686

3. Alakuijala, J. and Szabadka, Z., *2016.* Brotli: Compressed Data Format. Internet Draft (IETF, July 2016). Defines the Brotli lossless compressed data format using LZ77 + Huffman coding.

4. Amaeer, M., Hasan, K.M.A., Mahmood, M.A., Shuvro, P. and Awal, A., 2019. An efficient compression scheme for Bangla natural text by 16-bit Unicode transformation. *Proceedings of the 2nd International Conference on Innovation in Engineering and Technology (ICIET)*, Dhaka, Bangladesh, pp. 1–6. DOI: 10.1109/ICIET48527.2019.9290703

5. Awajan, A. and Abu Jrai, E., 2015. Hybrid Technique for Arabic Text Compression. *Global Journal of Computer Science and Technology: C – Software & Data Engineering*, Vol. 15, Issue 1, pp. 1-6. Online ISSN 0975-4172; Print ISSN 0975-4350

6. Azeez, N.A. and Lasisi, A.A., 2016. Empirical and statistical evaluation of the effectiveness of four lossless data compression algorithms. *Nigerian Journal of Technological Development*, Vol. 13, Issue 2, pp. 64–73. DOI:10.63746/njtd.v13i2.5

7. Bařina, D. (2023). *Experimental lossless data compressor*. *Microprocessors and Microsystems, Vol. 98*, Issue 4, Article104803. https://DOI.org/10.1016/j.micpro.2023.104803

8. Barrios Y., Sánchez A., Guerra R. and Sarmiento R., 2021. Hardware implementation of the CCSDS 123.0 B-2 near lossless compression standard following an HLS design methodology. *Remote Sensing*, Vol.13, Issue 21, pp. 4388. https://DOI.org/10.3390/rs13214388.

9. Bartrina C, Bartrina-Rapesta C, Serra J, Magli E, Serra-Sagristà J. IoT–Edge–Cloud compression pipeline with AES-128/HMAC for secure and energy-efficient data transmission. IEEE Internet Things J. 2017; Vol. 4, Issue 2:509-518. DOI:10.1109/JIOT.2016.2638338.

10. Bartrina-Rapesta, J., Blanes, I., Aulí-Llinàs, F., Serra-Sagristà, J., Sánchez Silva, V., and Marcellin, M. W., 2017. A Lightweight Contextual Arithmetic Coder for On-Board Remote Sensing Data Compression. *IEEE Transactions on Geoscience and Remote Sensing*, Vol.55, No. 8, pp. 4825-4835. DOI:10.1109/TGRS.2017.2701837

11. Battini, J., Bhatt, C.N., Krishna Reddy, C.V. and Sivani, K., 2014**.** FPGA implementation of secure image compression with 2D-DCT using Verilog HDL. *In: Proceedings of the 2014 2nd International Conference on Devices, Circuits & Systems*

(ICDCS'14), Karunya University, Coimbatore, India, March 2014, pp. 1-4. DOI:10.1109/ICDCSyst.2014.6926210.

12. Bhattacharjee, A. K., 2013. Comparison study of lossless data compression algorithms for text data. *IOSR Journal of Computer Engineering*, Vol. 11, No. 6, pp. 15-19. DOI: 10.9790/0661-1161519

13. Blanes, I., Magli, E. and Serra-Sagristà, J., 2014. *A Tutorial on Image Compression for Optical Space Imaging Systems*. IEEE Geoscience and Remote Sensing Magazine, Vol. 2, No. 3, pp. 8-26. DOI: 10.1109/MGRS.2014.2352465.

14. Blunier, I., Magli, E., and Serra-Sagristà, J.,2014. *A tutorial on image compression for optical space imaging systems. IEEE Geoscience & Remote Sensing Magazine*, Vol. 2, no. 3, pp. 8-26, Sept., DOI: 10.1109/MGRS.2014.2352465.

15. Boopathiraja, S. and Kalavathi, P., 2018. A hybrid lossless encoding method for compressing multispectral images using LZW and arithmetic coding. *International Journal of Computer Sciences & Engineering*, Vol. 6, Special Issue-4, pp. 313–318.

16. Boutell, T., 1997. PNG (Portable Network Graphics) Specification Version 1.0. Technical Report. RFC 2083. IETF. IETF Datatracker+2Semantic Scholar+2

17. Bruni, G. and Johansson, H.T., 2020. DPTC-An FPGA based trace compression. *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 67, Issue 1, pp. 189–197. https://DOI.org/10.1109/TCSI.2019.2945179.

18. Burrows, M. and Wheeler, D. J. , 1994. *A Block-Sorting Lossless Data Compression Algorithm*. SRC Research Report 124, Digital Equipment Corporation, Palo Alto, California, USA. Available at: https://www.cs.jhu.edu/~langmea/resources/burrows_wheeler.pdf

19. Butta Singh, A., Kaur, A., and Singh, J. (2015). *A review of ECG data compression techniques. Int. J. Comput. Appl.*, Vol.116, Issue 11, pp. 39–44.DOI: 10.5120/20384-2644

20. Carvalho, DR. Seznec, A., 2021. Understanding cache compression. *ACM Transactions on Architecture and Code Optimization*, Vol. 18, Issue 3, pp.1-27. DOI:10.1145/3457207.

21. Chang, Y. and Sobelman, G. E., 2024. Lightweight lossy/lossless ECG compression for medical iot systems, *IEEE IoT Journal*, Vol. 11, Issue 7, pp. 12450–12458, DOI: 10.1109/JIOT.2023.3336995.

22. Collet, J. L. (2013). *LZ4 – Extremely fast compression*. Available at: https://github.com/lz4/lz4

23. Collet, Y. (2016). *Zstandard compression*. Available at: https://github.com/facebook/zstd

24. CompuServe Inc., 1987. Graphics Interchange Format (GIF) specification, Version 87a. CompuServe Information Service, Columbus, OH, USA. Available at: https://www.w3.org/Graphics/GIF/spec-gif87.txt

25. Consultative Committee for Space Data Systems (CCSDS), 2005. CCSDS 122.0-B-1: Recommended Standard Image Data Compression. NASA/CCSDS. November 2005.

26. Devetak, I. and Winter, A., 2003. Classical data compression with quantum side information. *Physical Review A*, Vol. 68, Issue 4, p. 042301. DOI: 10.1103/PhysRevA.68.042301.

27. Deutsch, P., 1996. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951. IETF. May 1996.

28. Dhawale, N., 2014. Implementation of Huffman algorithm and study for optimization. *In: Proceedings of the 2014 International Conference on Advances in Communication and Computing Technologies (ICACACT 2014)*. Mumbai, India, 2014, pp. 1-6, DOI: 10.1109/EIC.2015.7230711.

29. Ebraheem, N.Q. and Ali, M.S., 2021. Features of application of data compression methods. *Scientific Quarterly Refereed Journal, Lebanese French University – Erbil, Kurdistan Region, Iraq*, Vol. 6, Issue 3, pp. 969–983. ISSN 2518-6566 (Online); ISSN 2518-6558 (Print).

30. Fiergolla, S. and Wolf, P., 2021. Improving Run Length Encoding by Preprocessing. *In: Proceedings of the 31st Data Compression Conference (DCC 2021), Snowbird, UT, USA, 23–26 March 2021*, p. 341. DOI: 10.1109/DCC50243.2021.00051

31. Fitriya, L.A., Purboyo, T.W. and Prasasti, A.L., 2017. A review of data compression techniques. *International Journal of Applied Engineering Research*, Vol. 12, Issue 19, pp. 8956–8963

32. Food and Agriculture Organization of the United Nations (FAO). AGROVOC: Multilingual agricultural thesaurus [Internet]. Rome: FAO; [accessed 2025 Jul 23]. Available from: https://agrovoc.fao.org/browse/agrovoc/en/

33. Golomb, S.W., 1966. Run-length encodings (Golomb codes). *IEEE Transactions on Information Theory*, Vol. 12, Issue 3, pp. 399–401. DOI: 10.1109/TIT.1966.1053907.

34. Google. 2015. *Brotli*. Available at: https://github.com/google/brotli/releases/tag/v0.2.0

35. Guguloth, E., Vadtya, S., Kudithi, T., Shanmugam, M. R., and Banoth, A. N., 2025. FPGA implementation of high throughput encoder and decoder design of lossless canonical Huffman machine. *Results in Engineering*, Vol. 26, Art. no. 105037. DOI: 10.1016/j.rineng.2025.105037

36. Guo, J., Xie, R. and Jin, G., 2019. An efficient method for NMR data compression based on fast singular value decomposition. *IEEE Geoscience and Remote Sensing Letters*, early access. https://DOI.org/10.1109/LGRS.2018.2872111.

37. Gündoğar, Z., Töreyin, B. U., and Demiralp, M., 2018, Tridiagonal Folmat Enhanced Multivariance Products Representation Based Hyperspectral Data Compression,

in *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 9, pp. 3272-3278, DOI: 10.1109/JSTARS.2018.2851368.

38. Gupta, A., Bansal, A., and Khanduja, V., 2017. Modern lossless compression techniques: Review, comparison and analysis. In *Proceedings of the 2nd IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, Coimbatore, India, Vol. 3. https://DOI.org/10.1109/ICECCT.2017.8117850

39. Gupta, S., Yadav, A., Yadav, D. et al, 2022. A scalable approach for index compression using wavelet tree and LZW. *International Journal Information Technology*, Vol. 14, pp. 2191–2204. https://DOI.org/10.10 07/s4 1870-022-00915

40. Hameed, M., Shakor, H. and Razak, I., 2018. Low power text compression for Huffman coding using Altera FPGA with power management controller. *In: Proceedings of the 2018 1st International Scientific Conference of Engineering Sciences (ISCES), Iraq , pp. 18-23, DOI: 10.1109/ISCES.2018.8340521*

41. Hanoune, M. and Lysmos, M.E.G., 2020. Data Compression Mechanisms in an Intelligent E-Health Gateway for Medical Monitoring Applications. *Procedia Computer Science*, Vol. 175, pp. 578–584. DOI: 10.1016/j.procs.2020.07.083

42. Hanumanthaiah A., Gopinath C., Arun B., Hariharan, R., and Murugan, R. ,2019. Comparison of lossless data compression techniques in low-cost low-power (LCLP) IoT systems. *Proceedings of the 9th International Symposium on Embedded Computing and System Design (ISED)*. Kollam, India, pp. 1–5. DOI: 10.1109/ISED48680.2019.9096229.

43. Holtz, K., 1993. The evolution of lossless data compression techniques. *Proceedings of WESCON '93*, San Francisco, CA, USA, pp. 140–145. DOI: 10.1109/WESCON.1993.488424.

44. Hosseini, M. (2012). *A survey of data compression algorithms and their applications. Applied Advanced Algorithms*, pp. 1–14

45. Huffman, D.A., 1952. A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, Vol. 40, Issue 9, pp. 1098–1101.

46. Hunter, R. and Robinson, A.H., 1980. International digital facsimile coding standards. *Proceedings of the IEEE*, Vol. 68, Issue 7, pp. 854–867. DOI: 10.1109/PROC.1980.11751.

47. Hwang, J., Dahir, N., Sarukkai, M. and Wright, G., 2023. *Curating Training Data for Reliable Large-Scale Visual Data Analysis: Lessons from Identifying Trash in Street View Imagery*. Sociological Methods & Research, 52(3), pp. 1155-1200. DOI:10.1177/00491241231171945. Available at: https://purl.stanford.edu/xq145bk1280

48. Ignatoski, M., Lerga, J., Stanković, L. and Daković, M., 2020. Comparison of entropy and dictionary-based text compression in English, German, French, Italian, Czech,

Hungarian, Finnish, and Croatian. *Mathematics*, Vol. 8, Issue 7, Article 1059. https://DOI.org/10.3390/math8071059

49. Intel Berkeley Research Lab: Wireless Sensor Data Set. http://db.lcs.mit.edu/labdata/labdata.html

50. Jayasankar, U., Vengattaraman, V. and Ponnurangam, D., 2021**.** A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *Journal of King Saud University - Computer and Information Sciences*, Vol. 33, Issue 2, pp. 119–140. DOI: 10.1016/j.jksuci.2018.05.006

51. Jeevan B, Bhatt CN, Krishna CV, Sivani K. 2014. FPGA implementation of secure image compression with 2D-DCT using Verilog HDL. In: Proceedings of the 2nd International Conference on Devices, Circuits and Systems (ICDCS). Coimbatore, India. pp. 1–4. DOI: 10.1109/ICDCSyst.2014.6926210.

52. Jrai E.A., Alsharari S., Almazaydeh L., Elleithy K. and Abu-Hamdan O., 2023. Improving LZW compression of Unicode Arabic text using multi-level encoding and a variable-length phrase code. *IEEE Access*, 11, pp. 51915–51929.

53. Kamble, S. and Patil, S.B., 2015. FPGA based data compression using dictionary based LZW algorithm. *International Journal of Latest Trends in Engineering and Technology (IJLTET)*, Vol. 5, No. 4, pp. 104–109. ISSN 2278-621X.

54. Kavitha, P., 2016. *A survey on lossless and lossy data compression methods.* International Journal of Computer Science & Engineering Technology, Vol. 7, Issue 3, pp. 110–114. ISSN 2229-3345.

55. Ketshabetswe, K.L., Zungeru, A.M., Mtengi B., Lebekwe C.K. and Prabaharan S.R.S., 2021. Data compression algorithms for wireless sensor networks: A review and comparison. *IEEE Access*, Vol. 9, pp. 136872–136891. https://DOI.org/10.1109/ACCESS.2021.3116311

56. Khelifati, A., Khayati, M. and Cudré Mauroux, P., 2019. CORAD: Correlation aware compression of massive time series using sparse dictionary coding. *Proceedings of the IEEE International Conference on Big Data*, pp. 2289–2298.

57. Kiely, A., Xu, M., Song, W-Z., Huang, R. and Shirazi, B., 2010. Adaptive linear filtering compression on realtime sensor networks. *The Computer Journal*, Vol. 53 Issue 10, pp. 1606–1620. https://DOI.org/10.1093/comjnl/bxp128.

58. Kimura, N. and Latifi, S., 2005**.** A survey on data compression in wireless sensor networks. *In: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC 2005), Vol. 2*, pp. 8-13

59. Klein, S.T., Saadia, S. and Shapira, D., 2019**.** Better Than Optimal Huffman Coding. *In: Proceedings of the Data Compression Conference (DCC), Snowbird, UT, USA*, 2019, p. 582. DOI: 10.1109/DCC.2019.00094.

60. Kodituwakku, S. R. and Amarasinghe, U. S., 2010. Comparison of Lossless Data Compression Algorithms for Text Data. *Indian Journal of Computer Science and Engineering*, Vol. 1, No. 4, pp. 416-425.

61. Kolo, J. G., Shanmugam, S. A., Lim, D. W. G., Ang, L. M. & Seng, K. P., 2015. Fast and efficient lossless adaptive compression scheme for wireless sensor networks. *Computers & Electrical Engineering*, Vol. 41, pp. 275-287. DOI:10.1016/j.compeleceng.2014.06.008.

62. Lee, J., Hyun, E. and Jung, J.-Y., 2019 .A simple and efficient IQ data compression method based on latency, EVM, and compression ratio analysis.*IEEE Access*, Vol. 7, pp. 117436–117447.DOI: 10.1109/ACCESS.2019.2936218

63. Li, G., Peng, S., Wang, C., Niu, J., Yuan, Y. ,2019.An energy-efficient data collection scheme using denoising autoencoder in wireless sensor networks, in Tsinghua Science and Technology, Vol. 24, Issue 1, pp. 86–96, Feb. https:/ /DOI.or g/10.26 599/TpapST.2018.9010002

64. Li, H., Meng, W. and Zhang, X., 2014. A survey on recent approaches of mesh compressions. *Proceedings of the 2014 International Conference on Virtual Reality and Visualization*, Shenyang, China, pp. 50–57. DOI: 10.1109/ICVRV.2014.2.

65. Liang, Y. and Li, Yimei, 2014. An Efficient and Robust Data Compression Algorithm in Wireless Sensor Networks. *IEEE Communications Letters*, Vol. 18, No. 3, pp. 439-442. DOI:10.1109/LCOMM.2014.011214.132319.

66. Lin, M.-B., Lee, J.-F. and Jan, G. E., 2006. A Lossless Data Compression and Decompression Algorithm and Its Hardware Architecture. *IEEE Transactions on Very Large-Scale Integration (VLSI) Systems*, Vol. 14, No. 9, pp. 925-936, September 2006. DOI:10.1109/TVLSI.2006.884045

67. Lin, J., 2019. A New Perspective on Improving the Lossless Compression Efficiency for Initially Acquired Images. *IEEE Access*, Vol. 7, pp. 144895-144906. DOI:10.1109/ACCESS.2019.2944658.

68. Liu, Y., Li, B., Zhang, Y. and Zhao, X., 2021. A Huffman based joint compression and encryption scheme for secure data storage using physical unclonable functions. *Electronics*, Vol. 10, Issue 11, pp. 1267. https://DOI.org/10.3390/electronics10111267.

69. Logeswaran, R., 2002. A prediction-based neural network scheme for lossless data compression. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 32, Issue 4, pp. 358–365. DOI: 10.1109/TSMCC.2002.806744.

70. Lu, J., Verma, N. and Jha, N. K., 2016.Compressed signal processing on Nyquist-sampled signals, *IEEE Trans Comput*, Vol. 65, Issue 11**,** pp. 3293–3303, DOI: 10.17577/IJERTV11IS040055.

71. Lungisani, B.A., Zungeru, A.M., Lebekwe, C.K. and Yahya A., 2024. Autoencoder based image compression for wireless sensor networks. *Scientific African*, Vol. 24, pp. e02159. https://DOI.org/10.1016/j.sciaf.2024.e02159.

72. Mahdavi, M., Edfors, O., Öwall, V. and Liu, L., 2019. A low latency FFT/IFFT architecture for massive MIMO systems utilizing OFDM guard bands. *IEEE*

Transactions on Circuits and Systems I: Regular Papers, Vol. 66, No. 7, pp. 2763-2774. DOI:10.1109/TCSI.2019.2896042

73. Malhotra, K. and Singh, A.P., 2021. Implementation of decision tree algorithm on FPGA devices. *International Journal of Artificial Intelligence*, Vol. 10,Issue 1, pp. 131–138. https://DOI.org/10.11591/ijai.v10.i1.pp131-138

74. Manuel, E.M., Pankajakshan, V., Mohan, M.T., 2023. Efficient strategies for signal aggregation in low-power wireless sensor networks with discrete transmission ranges, *IEEE Sensors Letters*, Vol. 7, Issue 3, pp. 1–4, March Art no. 7500304, https://doi.org/10.1109/LSENS.2023.3250432

75. Marcelloni, F. and Vecchio, M., 2008. A Simple Algorithm for Data Compression in Wireless Sensor Networks. *IEEE Communications Letters*, Vol. 12, No. 6, pp. 411-413. DOI:10.1109/LCOMM.2008.080300.

76. Mishra, S. and Singh, S., 2016. A survey paper on different data compression techniques. *Indian Journal of Applied Research*, Vol. 6, Issue 5, pp. 738–740. ISSN 2249-555X

77. Mo, Y., Qiu, Y., Liu, J. and Ling, Y., 2011. A data compression algorithm based on adaptive Huffman code for wireless sensor networks. *Proceedings of the International Conference on Intelligent Computation Technology and Automation (ICICTA)*, Vol. 1, pp. 26–29. DOI: 10.1109/ICICTA.2011.8.

78. Mehta, R., Lobiyal, D.K., 2017.Utility-based performance analysis of cross-layer design in multi-flow ad-hoc networks. *International Journal of Information and Technology*, Vol. 9, pp.377–387. https:/ /DOI.org/10.10 07/s4 1870-017-0040-9

79. Mishra, M., Gupta G.S. and Gui X., 2022. Investigation of energy cost of data compression algorithms in WSN for IoT applications. *Sensors*, 22(19), p. 7685. https://DOI.org/10.3390/s22197685.

80. Mittal S. and Vetter J.S., 2016. A survey of architectural approaches for data compression in cache and main memory systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 27,no. 5, pp. 1524–1536. https://DOI.org/10.1109/TPDS.2015.2435788

81. Mohamed, M. I., Wu, W. Y. and Moniri, M., 2018.Adaptive data compression for energy harvesting wireless sensor nodes, *Proceedings of International Conference on Network Sensing and Control* (IEEE), pp. 633–638, DOI: 10.1109/ICNSC.2013.6548812.

82. Nasif A, Othman ZA, Sani NS. 2021. The deep learning solutions on lossless compression methods for alleviating data load on IoT nodes in smart cities. Sensors. Vol. 21, Issue 12:4223. DOI: 10.3390/s21124223.

83. Nasif, A., Ali Othman, Z., Sani, N.S., Hasan, M.K. and Abudaqqa, Y., 2024**.** Huffman Deep Compression of Edge Node Data for Reducing IoT Network Traffic. *IEEE Access*, Vol. 12, pp. 122988–122997. DOI:10.1109/ACCESS.2024.3452669.

84. Oswald, C., and Sivaselvan, B. ,2018. Text and image compression based on a data mining perspective. *Data Science Journal*, 17, Article 12. https://DOI.org/10.5334/dsj-2018-012

85. Pandey M, Shrivastava S., Pandey. S. and Shridevi. S., 2020, . An enhanced data compression algorithm, *Proc Int Conf Emerging Trends Info Tech Eng (*IEEE), 1–4, DOI:10.1109/ic-ETITE47903.2020.223.

86. Patel, N., and Chaudhary, J.,2017. *Energy efficient WMSN using image compression: A survey.* In *Proc. Int'l Conf. Innovative Mechanisms for Industry Applications (ICIMIA)*, Bangalore, pp. 124–128.DOI: 10.1109/ICIMIA.2017.7975585.

87. Pavlov, I. (2002). *7-Zip*. Available at: http://www.7-zip.org/

88. Peng J., Kim C.-S. and Kuo C.-C.J., 2005. Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation*, 16(6), pp. 688–733. DOI: 10.1016/j.jvcir.2005.03.001.

89. Porwal, S., Chaudhary, Y., Joshi, J. and Jain, M., 2013**.** Data Compression Methodologies for Lossless Data and Comparison between Algorithms. *International Journal of Engineering Science and Innovative Technology (IJESIT)*, Vol. 2, Issue 2, March 2013, pp. 142–147. ISSN 2319-5967

90. Pushpalatha S., Shivaprakasha K.S., 2020. Data Compression for Wireless Sensor Networks: A Comparative Analysis, Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 2020, pp. 284– 286, https://DOI.org/10.11 09/I- SMAC49090.2020.9243522

91. Qiu T, Sun J, Chen N, Zhang S, Si W and Wang X, 2025.Olive-like networking: A uniformity driven robust topology generation scheme for IoT system, *IEEE Trans Comput*, 74(1) 86–100, DOI: 10.1109/TC.2024.3465934.

92. Rasheed M K, Padma T, Kumari C U and Rao N M, 2021. Lossless compression and implementation for medical signals using verilog, *Proc Int Conf Intell Comput Contr Syst* (IEEE). 1–6, DOI: 10.1109/ICICCS51141.2021.9432328.

93. Rebollo Neira L., 2019. Effective high compression of ECG signals at low level distortion. *Scientific Reports*, Vol. 9, No.1, p. 4564. https://DOI.org/10.1038/s41598-019-40350-x.

94. Rehman, M., Sharif, M., and Raza, M. ,2014. Image compression: A survey. Technology, Vol. 7, No. 4, pp. 656-672. DOI:10.19026/rjaset.7.303

95. Rekha, H. and Samundiswary P., 2015. Survey on low power adaptive image compression techniques for WSN. In: *2015 International Conference on Communications and Signal Processing (ICCSP)*, Melmaruvathur, India. IEEE, pp. 1128–1132. https://DOI.org/10.1109/ICCSP.2015.7322679

96. Rice, R.F., 1979. Some practical universal noiseless coding techniques. *JPL Technical Report 79-22*, Jet Propulsion Laboratory, Pasadena, California, USA

97. Rigler, S., Bishop, W. and Kennings, A.,2007. FPGA-Based Lossless Data Compression using Huffman and LZ77 Algorithms, *2007 Canadian Conference on Electrical and Computer Engineering*, Vancouver, BC, Canada, pp. 1235-1238, DOI: 10.1109/CCECE.2007.315.

98. Rissanen, J., 1983. A Universal Data Compression System, *IEEE Transactions on Information Theory*, vol. 29, no. 5, pp. 656–664, DOI: 10.1109/TIT.1983.1056714v

99. Roy N.R., Chandra P. ,2020. Energy dissipation model for wireless sensor networks: a survey. Int j inf Technology 12:1343–1353. https://doi.org/10.1007/s41870-019-00374-y

100. Sadler, C. M., and Martonosi, M. (2006). Data compression algorithms for energy-constrained devices in delay tolerant networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys '06)*, pp. 265–278. ACM. https://DOI.org/10.1145/1182807.1182834

101. Sadaghiani, A.K. and Ghanbari, M., 2019**.** An optimized hardware design for high-speed 2D-DCT processor based on modified Loeffler architecture. *In: Proceedings of the 2019 27th Iranian Conference on Electrical Engineering (ICEE 2019), Yazd, Iran*, pp. 1476-1480. DOI:10.1109/IranianCEE.2019.8786608.

102. Salomon, D., and Motta, G. (2006). *Data compression: The complete reference* (4th ed.). Springer. ISBN: 978-1-84628-602-5

103. Sanborn S, Ma X. 2005. Quantifying information content in data compression using the autocorrelation function. *IEEE Signal Processing Letters*, Vol. 12, Issue 3:230-233. DOI:10.1109/LSP.2004.842264.

104. Sarangi S. and Baas B., 2023. Energy-efficient canonical Huffman decoders on many-core processor arrays and FPGAs. *Integration: The VLSI Journal*, 88, pp. 156–165. DOI: 10.1016/j.vlsi.2022.09.015.

105. Sayood, K., 2012**.** *Introduction to data compression*. 4th ed. Burlington, MA: Morgan Kaufmann Publishers. ISBN 978-0-12-415796-5.

106. Scaglione, A. and Servetto, S.D., 2002. On the interdependence of routing and data compression in multi hop sensor networks, *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking (MobiCom 2002)*, Atlanta, GA, USA, September 2002, pp. 140–147. DOI: 10.1145/570662.570663.

107. Seward, J.,1996. *The bzip2 program*. Available at: http://www.bzip.org/

108. Shah, D. U. and Vithlani, C. H., 2011. *Efficient Implementations of Discrete Wavelet Transforms using FPGAs*. International Journal of Advances in Engineering and Technology, 1(4), pp. 100-111.

109. Shao Z., Di Z.-X., Feng Q., Wu Q., Fan Y., Yu X. and Wang W., 2022. A high throughput VLSI architecture design of canonical Huffman encoder. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(1), pp. 209–213. (Published online June 2021). DOI: 10.1109/TCSII.2021.3091611.

110. Shannon, C.E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, Vol. 27, Issue 3, 379–423 and 623–656. DOI: 10.1002/j.1538-7305.1948.tb01338.

111. Sharaff, A., Jain, M. and Modugula, G., 2022. Feature based cluster ranking approach for single document summarization. *International Journal of Information Technology,* Vol. 14, pp. 2057–2065. https://DOI.org/10.10 07/s4 1870-021-00853-1

112. Sharma K, Tiwari S, Gawande S S, (2022). Data compression and decompression method and system for communication system, *Int J Eng Res Technol*, 11(4) 85–88, DOI:10.17577/IJERTV11IS040055.

113. Shoaib, M., Jha, N.K. and Verma. N,2015. Signal processing with direct computations on compressively sensed data, *IEEE Trans Very Large Scale Integr Syst*, Vol.223, Issue 1, pp.30–43, https://DOI.org/10.29304/jqcm.2022.14.1.904.

114. Simrandeep, K. and Sulochana, P., 2012. A 5-Bit CAM-Based Dictionary for LZW Compression. *International Journal of information science and techniques, Vol* 2, Issue 4, pp. 87–95., https://DOI.org/10.5121/ijist.2012.2407

115. Singh, A.K., 2019. A wireless networks flexible adoptive modulation and coding technique in advanced 4G LTE. International Journal of Information Technology. Vol.11, pp. 55–66. https://DOI.org/10.10 07/s4 1870-018-0173-5

116. Sridevi, S., Vijayakumar, R., and Anuja, R. 2012. A survey on various compression methods for medical images. *International Journal of Intelligent Systems and Applications*, Vol.4, No.3,pp.13–19. DOI: https://DOI.org/10.5815/ijisa.2012.03.02

117. Srisooksai, T., Keamarungsi, K., Lamsrichan, P., and Araki, K., 2012. *Practical data compression in wireless sensor networks: A survey.* Journal of Network and Computer Applications, Vol. 35, Issue 1, pp. 37-59.

118. Smith, C.A., 2010. A survey of various data compression techniques. *International Journal of Recent Technology and Engineering*, Vol. 2, pp. 1–20.

119. Su, Y., Lu, X., Huang, L., and Guizani, M., 2019. A novel DCT-based compression scheme for 5G vehicular networks. *IEEE Transactions on Vehicular Technology*, Vol. 68, no. 11, pp. 10872-10881, Nov. 2019, DOI: 10.1109/TVT.2019.2939619.

120. Suarjaya, I.M.A.D., 2012. A new algorithm for data compression optimization. *International Journal of Advanced Computer Science and Applications (IJACSA)*, Vol. 3, Issue 8, pp. 14–17. http://dx.DOI.org/10.14569/IJACSA.2012.030803.

121. Subramanian, B., Palanisamy, K. and Surya Prasath V.B., 2021. On a hybrid lossless compression technique for three-dimensional medical images. *Journal of Applied Clinical Medical Physics*,Vol. 22, Issue 5, pp. 274–283. https://DOI.org/10.1002/acm2.12960.

122. Sudarshan, A. Health Care Analytics dataset. Kaggle. Available at: https://www.kaggle.com/datasets/abisheksudarshan/health-care-analytics (Accessed April 23, 2023)

123. Sudhakar, R., Karthiga, M.R. and Jayaraman, S., 2005. Image compression using coding of wavelet coefficients – a survey. *ICGST-GVIP*, Vol. 5, pp. 25–38.

124. Tan L., Jiang J. and Zhang Y., 2010. Bit-error aware lossless compression of waveform data. *IEEE Signal Processing Letters*, Vol. 17, Issue 6, pp. 547–550. DOI: 10.1109/LSP.2010.2046696.

125. Tariq, J., Farhan, M. and Abdulameer, M., 2022. Improve Compression Data Using Serial Combination Techniques. *In: Proceedings of the 8th International Conference on Contemporary Information Technology and Mathematics (ICCITM 2022), Mosul, Iraq*, pp. 290-293. DOI:10.1109/ICCITM56801.2022.

126. Taubman, D.S. and Marcellin, M.W., 2002. *JPEG2000: Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, Dordrecht. DOI:10.1007/978-1-4615-0799-4

127. Teja, G. R., Sruthi, R. S., Tomar, K. S., Sivanantham, S. and Sivasankaran, K., 2015. Verilog implementation of fully pipelined and multiplier-less 2D DCT/IDCT JPEG Architecture. *Online International Conference on Green Engineering and Technologies (IC-GET 2015)*, Coimbatore, India,pp. 1-5, DOI: 10.1109/GET.2015.7453819

128. Thomas, S. and Mathew, T., 2016. Lossless address data compression using quadtree clustering of the sensors in a grid-based WSN. *Ad Hoc Networks*,Vol. 56, pp. 35–47. https://DOI.org/10.1016/j.adhoc.2016.11.015

129. Tosoni, F., Bille, P., Brunacci, V., Angelis, A.D., Ferragina, P. and Manzini, G., 2025.Toward greener matrix operations by lossless compressed formats, *IEEE Access*, Vol.13, pp. 56756–56779, DOI: 10.1109/ACCESS.2025.3555119.

130. Trakimas, M., and Sonkusale, S.R., 2011. An adaptive resolution asynchronous ADC architecture for data compression in energy constrained sensing applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol.58,Issue 5, pp. 921–934. https://DOI.org/10.1109/TCSI.2010.2092132.

131. Usama, M., Malluhi, Q.M., Zakaria, N., 2021. An efficient secure data compression technique based on chaos and adaptive Huffman coding. *Peer-to-Peer Networking and Applications*, vol.14, pp. 2651–2664. https://DOI.org/10.1007/s12083-020-00981-8.

132. Verma K., 2021. *Hindi Bible Dataset*. Kaggle. Available at: https://www.kaggle.com/datasets/kapilverma/hindi-bible [Accessed 2 Oct. 2024].

133. Vito, S., 2008. Air Quality [Dataset]. *UCI Machine Learning Repository*. https://DOI.org/10.24432/C59K5F

134. Welch, T.A., 1984. A technique for high-performance data compression. *Computer*, Vol. 17, Issue. 6, pp. 8–19. DOI: 10.1109/MC.1984.1659158.

135. Witten, I.H., Neal, R.M. and Cleary, J.G., 1987. Arithmetic coding for data compression. *Communications of the ACM*, Vol. 30, Issue 6, pp. 520–540. DOI: 10.1145/214762.214771.

136. *Wu, C.H. and Tseng, Y.-C., 2011.* Data Compression by Temporal and Spatial Correlations in a Body-Area Sensor Network: A Case Study in Pilates Motion Recognition. *IEEE Transactions on Mobile Computing*, Vol. 10, No. 10, pp. 1459-1472. DOI:10.1109/TMC.2010.264v

137. Xu, X. and Zhou, Y., 2010. *Design of Image Data Compression IP Core Based on Processor Local Bus*. In: Proceedings of the 2nd International Workshop on Database Technology and Applications (DBTA), Wuhan, China. pp. 1-4. DOI: 10.1109/DBTA.2010.5658987

138. Yan W, Wu Z, Wang K, Zhou X, Zhang Q, Wang H. 2021. A resource-efficient, robust QRS detector using data compression and time-sharing architecture. In: IEEE International Symposium on Circuits and Systems (ISCAS). Daegu, Korea (South): IEEE; pp. 1–5. DOI: 10.1109/ISCAS51556.2021.9401523.

139. ZainEldin, H., Elhosseini, M.A. and Ali, H.A., 2015. Image compression algorithms in wireless multimedia sensor networks: A survey. *Ain Shams Engineering Journal*, Vol. 6, Issue 2, pp. 481–490. https://DOI.org/10.1016/j.asej.2014.11.001.

140. Zhao, L. L., Shao, X. X., Wu, M., Jiang, X. S. and Lu, L. Y. (2018). *Effects of Different Substrates and Particle Sizes on Wastewater Purification.* Environmental Science, 39, 4236–4241. DOI: 10.13227/j.hjkx.201712009scirp.org.

141. Zheng D, Xiu W, Ye L. 2021. Wireless sensor network data compression sampling technology based on unbalanced data collaborative filtering. In: *Proceedings of the International Conference on Computer, Engineering and Application (ICCEA)*, Kunming, China. 2021; pp. 194–199. DOI:10.1109/ICCEA53728.2021.00046.

142. Ziv, J. and Lempel, A., 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, Vol. 23, Issue 3, pp. 337–343. DOI: 10.1109/TIT.1977.1055714

143. Ziv, J. and Lempel, A., 1978. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, Vol. 24, Issue 5, pp.530–536. DOI: 10.1109/TIT.1978.1055934

APPENDIX I

Detail Results for Hybrid Compression Approach

Table I.1: Performance of Independent Compression Algorithms

Independent Algorithm	Data Type	Compression Ratio	Compression Speed (MB/s)	Decompression Speed (MB/s)	Efficiency Score
Pure LZMA	SD	3.60	35.26	281.19	0.4839
Pure LZMA	MD	3.74	31.78	456.98	0.4010
Pure LZMA	LD	3.79	30.99	524.70	0.2438
Pure Zstd	SD	4.45	45.99	141.30	0.5406
Pure Zstd	MD	4.59	76.61	389.41	0.6293
Pure Zstd	LD	94.49	546.22	593.79	0.6836
Pure Brotli	SD	4.60	7.15	132.01	0.2847
Pure Brotli	MD	5.00	38.38	259.61	0.4370
Pure Brotli	LD	117.11	312.66	426.47	0.6073
Pure Bzip2	SD	6.57	6.14	4.94	0.4236
Pure Bzip2	MD	7.56	20.18	4.95	0.4717
Pure Bzip2	LD	9.77	18.75	4.88	0.0210
Pure LZ4HC	SD	3.60	28.24	142.08	0.3047
Pure LZ4HC	MD	3.75	32.94	443.91	0.3986
Pure LZ4HC	LD	3.79	32.51	554.34	0.2580

Table I.2: Performance Analysis of Hybrid Compression Algorithms with LZMA Encoding

Hybrid Algorithm	Data Type	Compression Ratio	Compression Speed (MB/s)	Decompression Speed (MB/s)	Efficiency Score
LZMA + Zstd	SD	5.43	2.72	56.56	0.2952
LZMA + Zstd	MD	6.51	2.85	125.85	0.3673
LZMA + Zstd	LD	139.79	6.03	383.77	0.5670
LZMA + Brotli	SD	5.44	3.45	83.51	0.3266
LZMA + Brotli	MD	6.51	2.46	120.12	0.3622
LZMA + Brotli	LD	141.59	6.09	367.82	0.5650
LZMA + Bzip2	SD	5.34	3.43	56.94	0.2879
LZMA + Bzip2	MD	6.47	2.72	75.42	0.3310
LZMA + Bzip2	LD	139.77	5.82	349.25	0.5520
LZMA + LZ4HC	SD	5.43	3.64	89.37	0.3324
LZMA + LZ4HC	MD	6.51	2.71	121.80	0.3640
LZMA + LZ4HC	LD	141.10	6.01	376.25	0.5678

Table I.3: Performance Analysis of Hybrid Compression Algorithms with Zstd Encoding

Hybrid Algorithm	Data Type	Compression Ratio	Compression Speed (MB/s)	Decompression Speed (MB/s)	Efficiency Score
Zstd + LZMA	SD	4.44	13.79	114.95	0.2928
Zstd + LZMA	MD	4.59	21.10	343.99	0.3760
Zstd + LZMA	LD	94.89	382.25	627.83	0.6539
Zstd + Brotli	SD	4.45	35.77	167.57	0.4951
Zstd + Brotli	MD	4.59	70.34	354.94	0.5820
Zstd + Brotli	LD	94.49	1071.57	537.24	0.8057
Zstd + Bzip2	SD	4.39	25.89	87.68	0.3441
Zstd + Bzip2	MD	4.56	34.48	91.58	0.2690
Zstd + Bzip2	LD	94.30	623.42	534.89	0.6779
Zstd + LZ4HC	SD	4.45	46.31	283.02	0.6764
Zstd + LZ4HC	MD	4.59	60.79	393.64	0.5678
Zstd + LZ4HC	LD	94.82	1055.69	663.78	0.8597

Table I.4: Performance Analysis of Hybrid Compression Algorithms with Brotli Encoding

Hybrid Algorithm	Data Type	Compression Ratio	Compression Speed (MB/s)	Decompression Speed (MB/s)	Efficiency Score
Brotli + LZMA	SD	4.59	8.33	113.69	0.2742
Brotli + LZMA	MD	5.00	12.15	287.89	0.3480
Brotli + LZMA	LD	117.50	208.44	431.69	0.5814
Brotli + Zstd	SD	4.60	30.56	235.89	0.5430
Brotli + Zstd	MD	5.00	36.70	311.11	0.4620
Brotli + Zstd	LD	117.10	491.50	450.03	0.6683
Brotli + Bzip2	SD	4.53	17.99	66.09	0.2879
Brotli + Bzip2	MD	4.97	23.47	88.16	0.2660
Brotli + Bzip2	LD	116.21	343.58	338.98	0.5730
Brotli + LZ4HC	SD	4.60	28.79	134.21	0.4357
Brotli + LZ4HC	MD	5.00	33.03	281.41	0.4283
Brotli + LZ4HC	LD	117.08	426.05	365.97	0.6116

Table I.5: Performance Analysis of Hybrid Compression Algorithms with LZ4HC Encoding

Hybrid Algorithm	Data Type	Compression Ratio	Compression Speed (MB/s)	Decompression Speed (MB/s)	Efficiency Score
LZ4HC + LZMA	SD	4.12	11.28	63.53	0.1840
LZ4HC + LZMA	MD	4.26	12.79	79.29	0.1427
LZ4HC + LZMA	LD	84.33	23.05	323.52	0.3838
LZ4HC + Zstd	SD	4.09	32.47	323.83	0.5708
LZ4HC + Zstd	MD	4.24	34.49	485.04	0.4820
LZ4HC + Zstd	LD	67.40	35.18	553.80	0.4428
LZ4HC + Bzip2	SD	3.92	19.37	65.30	0.2145
LZ4HC + Bzip2	MD	4.16	20.30	78.84	0.1624
LZ4HC + Bzip2	LD	13.55	21.52	107.54	0.0794
LZ4HC + Brotli	SD	4.10	24.40	196.05	0.3964
LZ4HC + Brotli	MD	4.24	31.38	290.75	0.3480
LZ4HC + Brotli	LD	84.03	33.52	521.60	0.4760

Table I.6: Performance Analysis of Hybrid Compression Algorithms with Bzip2 Encoding

Hybrid Algorithm	Data Type	Compression Ratio	Compression Speed (MB/s)	Decompression Speed (MB/s)	Efficiency Score
Bzip2 + LZMA	SD	6.55	8.26	51.06	0.4788
Bzip2 + LZMA	MD	7.55	10.59	47.39	0.4584
Bzip2 + LZMA	LD	9.80	13.30	48.63	0.0362
Bzip2 + Zstd	SD	6.57	19.68	46.18	0.5556
Bzip2 + Zstd	MD	7.55	19.26	43.17	0.4908
Bzip2 + Zstd	LD	9.76	18.97	49.97	0.0415
Bzip2 + Brotli	SD	6.57	17.47	53.96	0.5476
Bzip2 + Brotli	MD	7.56	18.89	43.73	0.4907
Bzip2 + Brotli	LD	9.77	17.38	50.62	0.0410
Bzip2 + LZ4HC	SD	6.56	19.55	52.37	0.5591
Bzip2 + LZ4HC	MD	7.55	18.55	46.22	0.4899
Bzip2 + LZ4HC	LD	9.76	18.09	40.97	0.0372

Appendix–II

SUPPLEMENTARY FIGURES FOR AI-GUIDED COMPRESSION MODULES

The Fig.I.1 below illustrates the internal structure of a Decision Tree derived from the Random Forest classifier trained on Wireless Sensor Network (WSN) signal data. It shows how the model classifies sensor signal windows into distinct categories—smooth, mixed, periodic, drift, burst, and noisy—based on key statistical features such as maximum difference, standard deviation, and entropy. This visualization corresponds to the AI-based classification process described in Section 9.5.2.

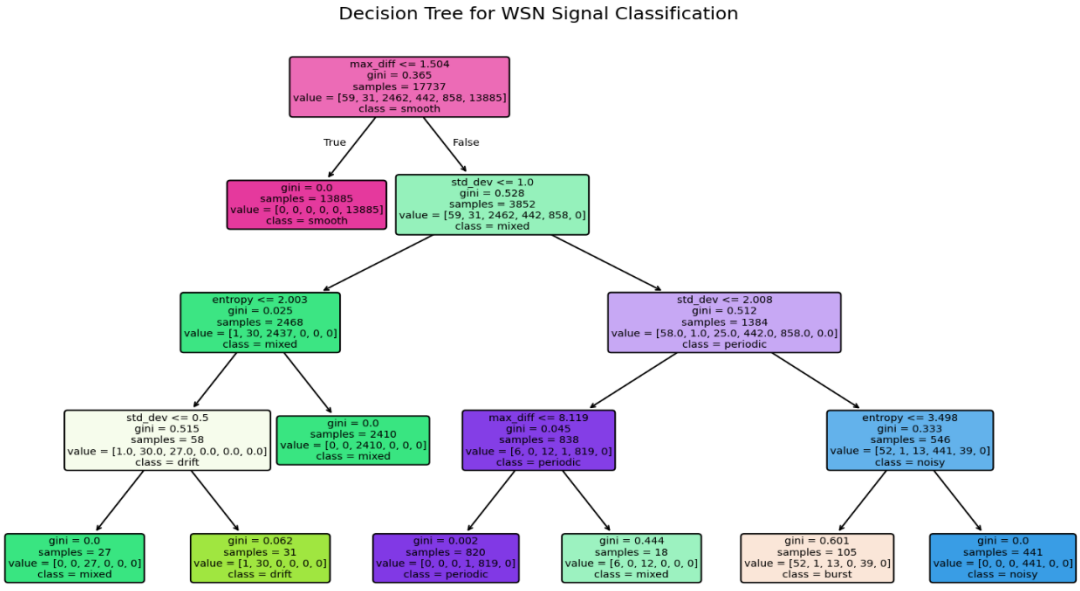

Fig. II.1: Decision Tree for WSN Signal Classification

AUTHOR'S BIOGRAPHY

Mukesh Sahu

Mukesh Sahu is a doctoral researcher in **data compression and efficient computing systems** at **Delhi Technological University (DTU), New Delhi, India**. His research centres on developing **memory-efficient and energy-aware lossless compression techniques** tailored for **resource-constrained environments** such as Wireless Sensor Networks (WSNs) and Internet of Things (IoT) systems. Building on advanced residual transformation and adaptive coding methodologies, his work emphasizes practical utility in low-power embedded systems and hardware-aware algorithm design. Mukesh's contributions include the design and evaluation of hybrid compression frameworks and AI-assisted signal processing models, validated on both synthetic and real-world datasets. He is actively preparing research publications for high-impact engineering conferences and journals, demonstrating strong foundations in systems design, algorithm optimization, and data-centric research methodologies.