e-ISSN: 2583-3472
Technoarete Transactions on Advances in Computer Applications (TTACA)
Volume 3, Issue 3, September 2024

Enhancing Data Security in Knowledge Databases: A Novel Integration of Fast Huffman Encoding and Encryption Techniques

Babita Kumari[1*], Dr. Neeraj Kumar Kamal[2], Dr. Arif Mohammad Sattar[3], Mritunjay Kr. Ranjan[4*], Rishu Jain[5]

[1]Research Scholar, P.G. Dept. of Mathematics and Computer Science, Magadh University, Bodh Gaya, India
[2]Assistant Professor, Dept. of Physics, A.M. College, Gaya, India
[3]Associate Professor, Dept. of MCA, Dewan Institute of Management Studies, Meerut, India
[4]Assistant Professor, School of Computer Sciences and Engineering, Sandip University, Nasik, India
[5]Assistant Professor, Dewan Institute of Management Studies, Meerut, India
*Corresponding Author Email: mcababitapaswan@gmail.com

Abstract

In today's rapidly changing technological environment, it is critical to protect knowledge databases, which are vast and elaborate. Conventional security techniques sometimes fail to manage speed, security, and data structures well at the same time. This paper proposes an improved model for improving data security in knowledge databases using the fast Huffman encoding algorithm. Huffman encoding not only shrinks the size of data but also hides some patterns to make it more difficult for unauthorized persons to access. In combination with encryption, the proposed approach provides both efficacy and strong security guarantees. The model is also capable of overcoming some of the major drawbacks of the conventional methods and enhancing the encryption capability and security of data. This paper describes the theoretical foundation of the model, how the model can be put into practice, and the proof of its efficiency. Studies show that it is much more secure and efficient than traditional approaches, as scientific studies prove. It has been seen that the proposed approach helps in minimizing the security issues and, at the same time, helps in managing the resources in a better way, which is a big boon in today's world where huge volumes of data are being generated on a daily basis.

Keywords

Data Security, Knowledge Databases, Huffman Encoding, Data Compression, Encryption, Efficiency, Security Framework, Information Protection

INTRODUCTION

In today's world, where technology developments are accelerating and the amount of data is increasing more than exponentially, it has become a critical issue for organizations to secure knowledge databases. The use of knowledge databases, which include large and complex databases of linked data and information, ontologies, and metadata for strategic planning and decision-making. These databases are relatively complicated and, thus, susceptible to hacking and other cyber threats [1]. Historically, techniques of securing data did not do justice to defining a balance of productivity as well as security, especially with a rise in more complex attacks. The current digital security solutions must encompass data and information accuracy, availability, and functionality. Most classical security techniques operate independent of each other, which leads to more risks and ineffective solutions, such as encryption or data compression. Saying that, encryption insulates data from sharing and unauthorized access. However, this process may be complex, especially in big data applications. Even though data compression aims at minimizing the extent of the file, it has the effect of disclosing data [2]. The problem that arises is a security framework that aligns these processes in order to facilitate a comprehensive knowledge database solution. This paper presents a new model that incorporates Huffman encoding, an appreciated compressing algorithm, into an overall encrypting system to protect knowledge databases. In this paper, the proposed model employs the Huffman encoding technique to minimize data size and conceal the patterns exploitable by attackers. After the compression, the data is encrypted, making it very secure [3]. This dual approach minimizes the size of the data set and enhances the rate of encrypting the information, making it less vulnerable to insecurity. This model is founded on security facets of data and their compression. The framework for the proposed coalesced index uses Huffman encoding, which optimally encodes symbols depending on their probability of occurrence. The extension with Huffman encoding enhances the encryption algorithms such as AES by minimizing the data size [4]. This integration enhances the approaches of compression and encryption to enhance the security of the knowledge databases. Details of the implementation of the proposed model and experimental results on simulated knowledge databases containing various types of data will be discussed in this paper. The conceptual experiences measure compression ratios, encryption times, decryption times, and data security. These metrics are examined in this research to demonstrate the enhanced security and efficiency that result from using Huffman encoding together with encryption to present a realistic solution that could be utilized by organizations with valuable data to safeguard. The existing literature on knowledge database security will be reviewed in this paper in terms of data compression and encryption strategies and the proposed model for handling the difficulties created by intricate data sets. The given framework improves data security by including Huffman encoding [5] together with its intensive encryption. The findings of this research may contribute to the development of better security mechanisms for knowledge databases in a globalized environment where confidentiality of data is crucial.

Objective:

- To establish the Huffman encoding model that is integrated with encryption in order to strengthen security on the knowledge databases.
- To minimize the computational cost at the sacrifice of some aspects of data compression and encryption, maintaining efficiency and reliability.
- To hide patterns of data and avoid various threats, create reliable protection against access by unauthorized individuals to highly informative databases.

RELATED STUDY

Table 1. Comparative Study on Category Details and Key features and Challenges

Category	Description	Key Features	Challenges
Background [6]	Knowledge repositories are designed to contain and access information with dependent and independent connections and contexts. These demand specific approaches for handling their characteristic features and the manner in which they are used.	Complexity of the data relationship Contextual data storage.	The conventional approaches do not work best in dealing with structures. Require advanced solutions.
Definition and Purpose [7]	Knowledge databases store structured, semi-structured data as well as unstructured data. They facilitate decisions, knowledge and organizational learning.	Semantic data support and ontology querying of data.	Challenges in querying and managing semantic data.
Characteristics [8]	Deal with complex real world data relationships. Integrate related, multiple types of data. The last element is known as adapting to change in data and relationships.	Complexity Connected information operability for volatile data.	Demand specific procedures of storing and searching the information. Some of the factors that make it difficult to maintain adaptability.
Applications [9]	Applicable in healthcare, finance, education, and research to name but a few in expert systems, decision support systems, and the like. Improves the organization's competencies by offering up-to-date information on the field.	Expert systems Decision support systems Content management	Different for different industries as it depends on the specific nature of the business.
Importance of Security [10]	Consequences include monetary, legal, and image loss costs. Outcomes can be monetary, legal, and image loss outcome-an Outcomes can be contained monetary, legal, image loss costs. Classical cryptographic procedures are unable to overcome the challenges posed by knowledge databases' architectures.	Covers financial, legal and reputation management risk. Preserves hierarchical databases.	The strengths and the weaknesses of the proposed approach.
Challenges [10]	Efficiency vs. safety. It is clear that complex structures are vulnerable to attacks. Security solutions cannot afford to worsen the performance of a database.	Trade-off between privacy, speed, and protection; data-related challenges; and their impact on performance.	Problems with immediate search requests and the ability to provide sufficient productivity when used frequently.
Emerging Solutions [11]	Combining modern algorithms, for example, Huffman encoding and standard security solutions help create broad frameworks. It is clearly seen that their integration results in better performance and level of protection.	Huffman encoding data compression integrated with encryption.	Interoperability issues as well as compatibility within the existing organizational systems.
Relevance of Research [12], [13]	The following study focuses on the advancement in security models that use Huffman encoding and encryption while achieving optimal throughput and minimizing the impact on security on databases.	Combines data compression with encryption, focuses only on enhancing the performance as well as the security of the data.	Organizational Implementation Issues and High Performance Work Systems.

Table 2. Huffman Encoding - Mechanism, Advantages, Applications, Limitations, and Relevance to Data Security

Segment	Subsection	Description
Mechanism of the Algorithm [14]	Frequency Analysis	The algorithm starts with the calculation of the frequency of each character in the given dataset. Frequently used characters are given shorter codes than the less frequently used characters.
	Binary Tree Construction	From the frequency data, a binary tree is formed, in which each node is a character of the text or document. The tree that is created is constructed in a way that the most often occurring characters reside closer to the root, and therefore they will require lesser path (codes).
	Code Generation	Moving from one node to another, produces the binary strings for the characters, thus making it more compact saving on the total size of the data.
Advantages of Huffman Encoding [15]	Efficiency	The algorithm generally provides massive data compression, which finds applications across the spectrum including file compression and data transmission.
	Optimality	When the exact probability of each symbol is well understood, Huffman coding is considered optimal for lossless compression. This means that it avoids a longer expected code length.
	Simplicity	The algorithm is simple and is easy to implement. Therefore no complicated mathematical computations are needed, making it suitable for real world implementation.
Applications [16]	File Compression Formats	Huffman coding is implemented in common file formats such as ZIP, JPEG and PNG because when data storage and transmission occurs most of the time the least amount of space is needed.
	Data Transmission	It has been largely used to minimize bandwidth in communication processes and act as a major contribution to improved communication throughput.
	Encoding Schemes	Huffman encoding is then used in a number of encoding techniques. For example, in multimedia and text retrieval, in order to reduce the amount of space required for storage.
Limitations [17]	Static vs. Dynamic	It has been depicted, Huffman coding is most efficient in situations whereby the characters in the data set do not change frequently from time to time.
	Overhead	The requirement to preserve and convey the Huffman tree can pose overhead, especially when data is small, and the advantage of the compression achieved may be offset by the tree size.
Relevance to Data Security [18]	Data Security Integration	The combination of Huffman encoding with encryption techniques opens the way to further improving data protection in knowledge base databases.
	Efficiency and Security	Data compression done by Huffman can assist in hiding a pattern of traffic as well as less traffic that requires encryption, meaning efficiency is achieved.

PROBLEM STATEMENT

In the age of big data, it is a difficult problem to protect knowledge bases because of their intricate and voluminous information. In many cases, traditional security approaches do not provide sufficient protection and, at the same time, do not cope with the protection effectively for complex structures of data. Flaws in the techniques used to compress and encrypt data make these databases easy targets for hackers as well as greedy system users. This research also seeks to overcome those limitations since the proposed integrated model is composed of fast Huffman encoding and enhanced encryption mechanisms. The model also intends to improve the efficiency of data security through optimizing data reduction with lossless data compression and encryption to minimize computational burden and data patterns that could be security threats.

HUFFMAN ENCODING ALGORITHM

One of the most common data compression techniques employed is Huffman encoding, whereby newly assigned codes to symbols are determined by the occurrence frequency of these symbols. On this basis, the codes for frequently used symbols are shorter, whereas less often used symbols are assigned longer codes [13], [19]. It is also useful in shrinking the amount of data stored or transmitted, as the method will reduce the size of the data. The most famous Huffman encoding is quite efficient and used in some compression standards like JPEG and PNG ((Table.1).

Table 3. Algorithm Approach (Algorithm-1)

```
Input: Label map data
Output: dict, encoded Data
S1. Compute the frequency of each symbol:
      symbols, freqs ← count(data)

S2. Construct nodes based on symbols and their frequencies:
num, nodes ← constructNode(symbols, freqs)
S3. For i = 1 to num - 1 do:
      • Sort nodes in ascending order of frequency:
        nodes ← sort(nodes, freq)
      • Create a new node with the two smallest nodes:
        newNode ← constructNode(nodes[1], nodes[2])
      • Assign child nodes to the new node:
        newNode.left ← nodes[1]
        newNode.right ← nodes[2]
      • Update the nodes array:
        nodes ← [newNode, nodes[3:end]]

S4. Traverse the tree to generate the dictionary:
      dict ← traverse Tree(nodes[end])

S5. Encode the data using the dictionary:
      encodedData ← encode (data, dict)
End
```

Implementation Process

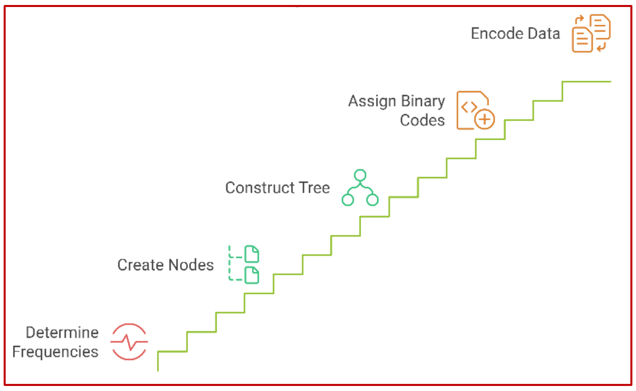


Figure 1. Data Encoding Process

The first procedure in the encoding of data is the determination of frequency densities of each sign in the input data (Figure 1). In this step, two things are made, namely, the list of images and the frequencies of the images. As it is an analogy with the symbol set of “ABACA,” we can state that the frequencies are {3, 1, 1} and the characters are {[A], [B], [C]}. With these symbols and frequencies, nodes are created that have areas for left and right child nodes, and each node has its own symbol and its frequency. Following that, a hierarchical tree is constructed from these pieces. The nodes are sorted by frequency again and again, where two less frequent words are combined into a new node. The new node is then assigned the value of the sum of the two nodes on the lower end of the frequency spectrum. They become its left and right children, respectively. This process goes on until we have only one node, and this is the tree root node. The next step is to go through the tree again and give them binary numbers to every symbol as shown below. Setting the left branches as `0` and the right branches as `1} builds a dictionary of the symbols to code. The last is to turn over the data map and change each of the symbols in line with their codes in binary. This means an identification, a brief, condensed, dual set of figures. The created product is the binary dictionary and the encoded data that is encoded in binary form, making it easier to store and send the data.

PROPOSED MODEL

For constructing an impenetrable security system for a resultant KB, the proposed model combines Huffman encoding for data compaction with new generation encryption algorithms. The foregoing resiliency means that to increase data safety and functionality, it is done in parallel to address issues that come with categorized data formats. In this way, for every non-reference pixel, the model defines the corresponding embeddable bits according to the label map that is generated. But when embedding and extracting the secret data reversibly, the label map has to be embedded in the embeddable bits. If performed directly, the label map takes away space that would otherwise be available for the actual secret data to be embedded. To avoid this, the model makes use of adaptive Huffman encoding for the label map such that it occupies the minimal space possible. In Huffman coding, a variable-length encoder, a set of the code words of the shortest length possible is developed by using probabilities of character occurrence to compress data without data loss. The process involves constructing a binary tree where characters occupy the leaf nodes, and branches represent encodings—typically "0" for one branch and "1" for the other. This encoding ensures efficient compression but does not produce unique results because:

- When two symbols are assigned the same probability in constructing the Huffman tree, the tree developed is none of them unique.
- It is important to realize that the branches 0 and 1 are not set during the encoding; it is possible to encode the left branch as 0 and the right branch as 0 too.

These two reasons make it possible to have different encoding outcomes. Nevertheless, the likely occurring characters are assigned shorter code words, whereas the less likely characters are given long code words. As in the proposed method, different images generate different label maps; a method of determining the probability of the elements in the label map to generate the shortest code words on average is a way of achieving better lossless compression (Figure 2).

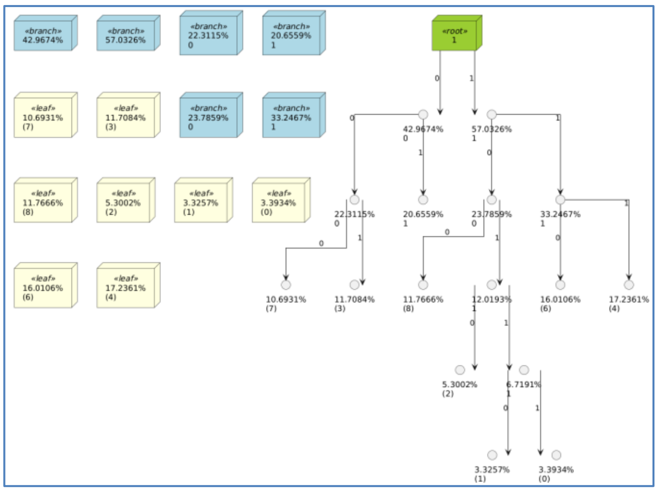


Figure 2. Huffman tree and label encoding results for Lena

In order to illustrate this process more comprehensively, the encoding process of the proposed method is described in this subsection with the Lena image [20]. First, we compute the frequencies of the nine labels 0-8 related to the label graph. According to the calculated frequencies, Huffman encoding is done for each label. Algorithm 1 provides a detailed account of the building process of the Huffman tree as follows. The generated label map is then converted into a binary sequence based on the mapping between the encoded label and the codeword [21]. From the above analysis, it can be known that the length of the binary sequence of the label map is able to be got by Eq. (i).

$$Len_t = \sum_{t=0}^{8}(num_t \cdot codeLen_t), \quad 0 \le t \le 8 \ \ldots\ldots \text{(i)}$$

Where,

- Num_t is the number of label t in the label graph and code.
- Len_t is the length of the code word corresponding to label t.

Principle of Rapid Huffman Encoding

Real-time JPEG encoding is possible if the time taken in each of the individual forms of operation corresponds with the rate at which the JPEG data is fed into the pipeline. In a standard JPEG encoding setup, where the relative sampling frequency is {2×2, 2×2, 2×2} for full sampling, each Minimum Coded Unit (MCU) consists of 64 pixels, requiring 64 clock cycles for loading. Common steps such as 2D-DCT, quantization, Zigzag scanning, and DPCM&RLE of the image generally require nearly 64 cycles each to be completed. But the duration of Huffman encoding is not determinate. In simple image encoding it can take less than 64 cycles, while in complex image encoding it can take more than 64 cycles. This results in some of the Huffman encoding stage being slower, sometimes slowing down the other modules and thereby not providing real-time encoding. Using relative encoding, Huffman encoding assigns short bit values to codes with more frequently used referenced data and longer codes for less frequently used reference data. For example, the given string ABAACDAAAB comprises the characters represented in (Table. 4), which shows the numerical representation of the frequency of each of those characters.

Table 4. Huffman Coding for Letters Based on Frequency

	A	B	C	D
Frequency	6	2	1	1
Huffman Code	0	10	110	111

The string of letters mentioned above consists of four letters distinct from each other, requiring 2 bits for representation; hence the total will be 20 bits for a 10-character sequence. Nevertheless, with an optimal Huffman coding from (Table. 4) below, the symbol sequence requires (6×1) + (2×2) + (1×3) + (1×3) = 16 bits, or 4 bits less. As a consequence of DPCM and RLE, multiple pairs of run length, size, and amplitude are produced [22]. When a Huffman table is used, the actual encoding process is easy; much simpler than EDS, because all the encoding is accomplished by looking up in a Huffman table to replace the actual data with the Huffman code. The Huffman encoding process relies on four specific tables: luminance AC coefficients, chrominance AC coefficients, luminance DC coefficients, and chrominance DC coefficients [23]. VLC and VLI are the two types of Huffman encoding. Different symbols are inferred by VLC for codewords of different lengths and can be determined from the Huffman table of (Run Length, Size). At the same time, VLI directly reflects the properties of the input data as an amplitude represented as a binary integer. For encoding, one has to substantially manipulate the splicing, shifting, and registering of VLC or VLI codes. To control this, a register bit pointer is assigned for every VLC and VLI codeword present within the stream. (Figure. 3) below shows the Huffman encoding process for each of the data pairs (Run Length, Size, Amplitude): A = 0; B = 10; C = 110; D = 1110; The major steps of Huffman encoding.

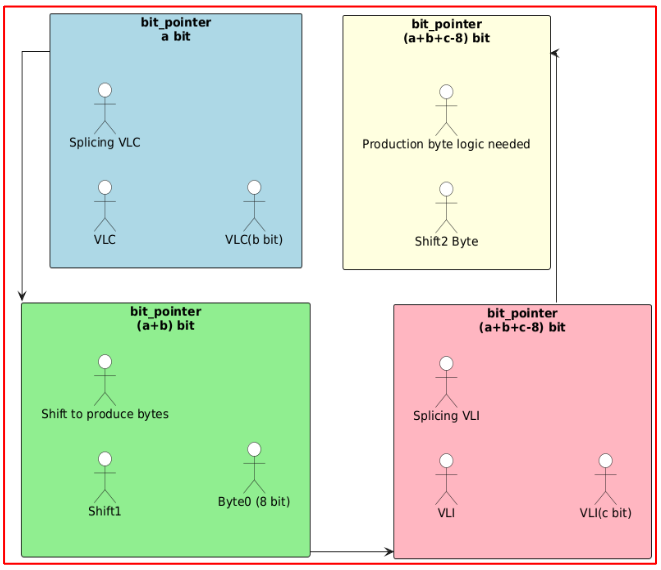


Figure 3. Runlength, Size, Amplitude) data in Huffman encoding

Bit pointer loads VLC (b bit) in the first clock cycle, and its effective number of bits is (a+b). Calculate the effective bits in the second clock cycle. In the third clock cycle, the bit pointer loads the VLI (c bit), and the effective number of bits is (a+b+c-8); in the fourth clock cycle, if (a+b+c-8) is greater than 8, generate the code-byte Byte1. Usually, Huffman encoding takes 4 clock cycles, but byte generation takes 2 if the effective bit number of the bit pointer is greater than 16 after splicing VLI or VLC. The entire process will take 4–6 clock cycles. Thus, this article optimizes (Figure. 3) and designs a Huffman encoding structure for double-byte splicing output to reduce clock cycles [24]. The above structure shows that VLI and VLC encoding can combine the splicing and shifting processes. Thus, one clock cycle can complete splicing and shifting. By increasing the bit pointer bit width, double-byte output occurs after splicing VLC and VLI since Byte 0 or Byte 1 output takes 1 to 2 cycles. Finally, Huffman encoding outputs double-byte data in a clock cycle, so the shift to produce bytes process takes one clock cycle. These two optimizations will reduce the time consumption of each (Run length, Size, Amplitude) in Huffman encoding to 2 clock cycles, but they won't handle complex situations. DPCM&RLE allows MCUs to output 64 pairs (Run length, Size, Amplitude). Huffman encoding with double byte splicing output processes 64 pairs (Run length, Size, Amplitude) in 128 clock cycles, twice as long as other JPEG modules. To achieve real-time encoding, limit the Huffman encoding cycle to 64 clock cycles so all modules in the pipeline spend the same cycle. Besides double-byte splicing output, this paper proposes a new Huffman encoding architecture (Figure 3). DCPM & RLE has two FIFOs for Run length, Size, and Amplitude (Figure 3) [25]. Different Huffman encoding units can read the data simultaneously because the first 32 groups are stored in FIFO32 and the rest in FIFO 64. Huffman0 and Huffman1 are operated simultaneously to reduce clock cycle consumption. Bit stream recombination is set up in the output section to sort and reassemble fragmented bit streams from different Huffman encoding units to recover them completely (Figure 4).

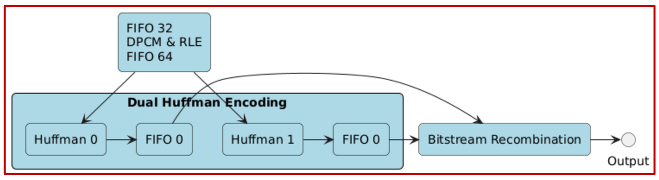


Figure 4. Huffman encoding architecture

The efficient Huffman encoding architecture, as illustrated in (Figure 4), is designed to deliver excellent performance and is structured into two main components: Functional Huffman Encoder (FHE) and bit stream recombination. This approach replaces the single-stage pipeline with a two-stage pipeline. Although the number of clock cycles needed to process a single DU is larger compared to processing it in a straightforward manner, this is compensated by more data throughput and processing capability. Thus, it becomes possible to encode such images that contain 64 pairs of Run Length, Size, and Amplitude values with the help of Huffman encoding for more than 64 clock cycles. Therefore, the elapsed time for each step involved in JPEG encoding is similar, which makes it possible for JPEG encoding to occur in a non-blocking fashion.

Dual Huffman encoding module

VLC encoding will assign some symbols shorter codewords than others, achieving at the end the least number of total bits utilized for encoding, thereby minimizing resource utilization. The Huffman encoding process can be broken down into three main stages: VLC encoding, VLI encoding, as well as doing the rest at the later stages to get the overall number of 16 bits, as illustrated in (Figure 5). The VLC code is developed after decoding all four Huffman tables at the same time, while the VLI code is directly extracted from the terminal input of a module. Due to the fact that the encoding involves the mapping of VLC and VLI codes which entails splicing, shifting, and registration onto the codeword register and sign register, a word_reg has been used here to record the codewords into a codeword register. Furthermore, a bit pointer register (bit_ptr) is used to count down the accurate number to the really effective bits used in the word_reg. The result of the Huffman coding is in double-byte form, and the maximum VLC length is 16 bits. To accommodate 12 bits for each of the input words, suitable for VLC and VLI encoding in one byte production cycle, the word_reg is set to have 32 bits, as shown below.

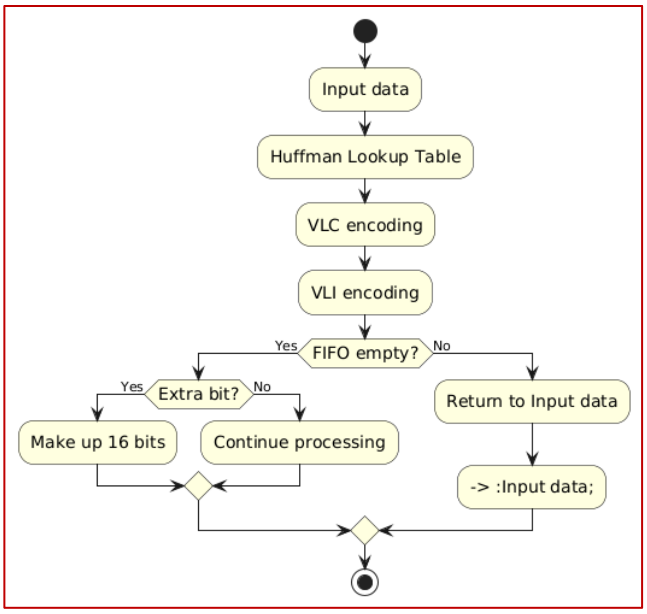


Figure 6. encoding process, VLI encoding process

In VLC encoding, the codeword generated from the lookup table is sliced and added to `word_reg`. The bit position within `word_reg` is determined based on the value stored in `bit_ptr`. If the value of `bit_ptr` exceeds 16 for one or both data segments, the excess bits are shifted to form a byte, which is then output. On the other hand, the VLC encoding requires a lookup table, although the VLI codeword is directly computed from the input and does not require a table lookup[26]. However, after processing a data unit (DU), the 'word_reg' may still contain valid bits. Finally, in the last step for making sure that the bit stream results in 16 bits, the least significant bits are filled with zeros depending on the number that is needed to get to 16. Simultaneously, the `end_reg` register is updated to record the effective bits of the lastest data. The information about the last byte is the necessary data required by the bitstream recombination module in the further step [27]. The architecture framework for both Huffman encoding methods is illustrated in Figure 4 below. It includes two Huffman encoding units and a single Huffman lookup table. The next table specifies the VLC codewords corresponding to each Run Length and Size values Figure 3 shows the lookup table for VLC codewords. The Huffman encoder processes both VLC and VLI codewords and stores the results in `FIFO0` and `FIFO1`. Alongside these results, valid signals (`Rd_en0` and `Rd_en1`) are used, as shown in (Figure 6), to determine the output for Huffman Codeword 0 and Huffman Codeword 1.

Bitstream Recombination

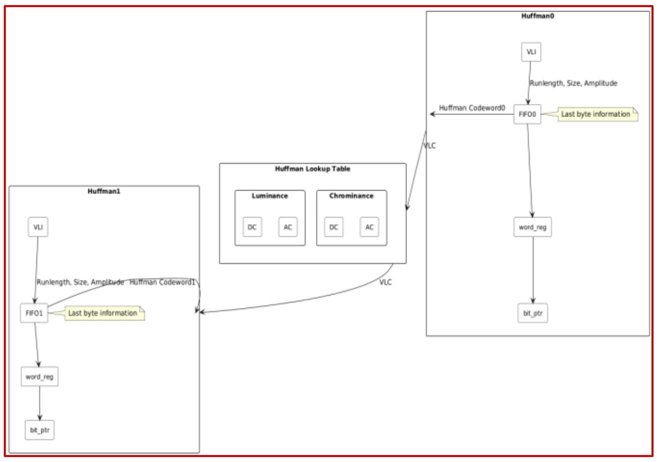


Figure 1. The structure of the dual Huffman encoding.

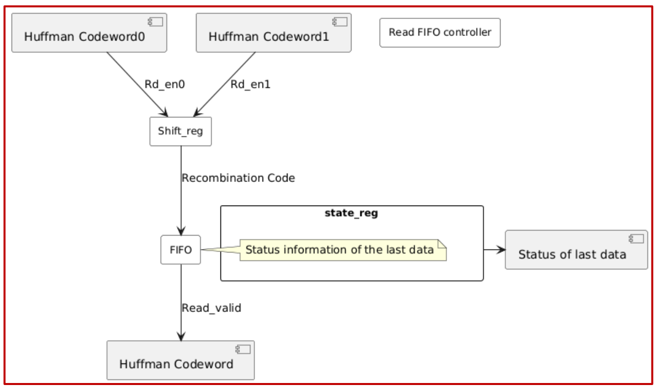


Figure 2. The bitstream recombination module

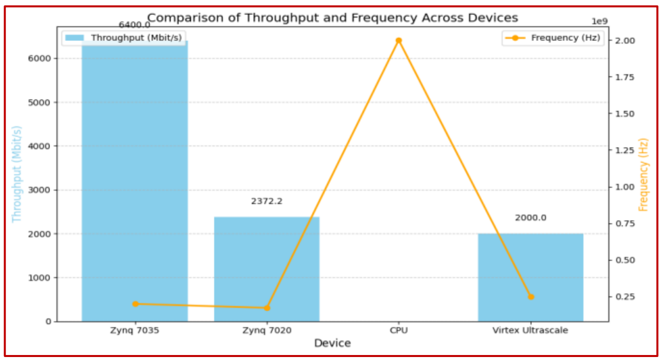
The (Figure. 8) shows that a shift register, termed as Shift_reg, is used for combining Huffman Code0 and Huffman Code1. For the purpose of requesting outputs from FIFO0 and FIFO1 as illustrated in Figure 5 above, the read FIFO controller module creates output signals known as Rd_en0 and Rd_en1. These read-valid signals are provided to the FIFOs to decode Huffman code words found in the Huffman tree. Every time the recombination occurs, the micro program begins to load simultaneously with the Shift_reg and the effective bit count becomes zero. For Huffman Codeword0, it simply outputs the value of the FIFO. In Huffman Codeword1 structure, the data is more often than not shifted as well as the output only merges. The next step after the Huffman encoding process is the position process: whether all data are combined or not, there must be an examination about whether there are still more valid bits remaining in Shift_reg or not. When any extra bits are there, the status information of the last data is available through State_reg.

RESULT AND DISCUSSION

The fast Huffman encoding method as described in this paper has also been tested on a hardware system. The simulation of the present design was carried out using the Xilinx Zynq7035 pcba, and the hardware design was implemented using the Xilinx Vivado 2017.2 tool. After providing the synthesis of Huffman encoding based on the proposed approach, a comparison of the results with related works is presented in (Table 5).

Table 5. Compare Huffman Encoding in Different Articles

Category / Device	Device	Frequency (Hz)	LUTs	Number of Cycles	Time (ns)	Throughput (Mbit/s)
Proposed (Zynq 7035) [28]	Zynq 7035	200M	624	64	320	6400
Zynq 7020 [29]	Zynq 7020	173M	761	448	2590	2372.2
CPU [30]	CPU	2G	--	256	--	--
Virtex Ultrascale [31]	Virtex Ultrascale	250M	42052	256	1024	2000



Graph 1. Comparison of Throughput (Mbit/s) and Frequency (Hz) Across Different Devices

As indicated in (Table. 5, Graph. 1), the resource consumption in Huffman encoding in other approaches is higher as compared to the fast Huffman encoding technique proposed herein. This is because existing approaches involve implementations on the Central Processing Unit (CPU), and these consume many clock cycles but take little processing time. Experiments shown in this paper show that the identified Huffman encoding maximally enhances throughput and processing time. As a result, the Huffman encoding method described in this work has the following benefits and achieves a non-blocking JPEG compression process as shown in (Table. 6).

Table 6. Overview of Compression, Encryption, and Decompression Processes

Section	**Subsection**	**Description**
Data Compression Using Huffman Encoding [32], [33]	Frequency Analysis	Determine how often each of the data elements such as text, numbers, pictures exists in the knowledge database. If you wish to find characters or symbols that appear most often in a given text, create a frequency table of the characters or symbols.
	Binary Tree Construction	Construct a binary tree of the frequency table generatedFromFile frequencies: Each node represents the record and the tree arrangement allows elements frequently to be seen closer to the root.
	Code Generation	Encourage preparation and construction of distinctive divergent binary string descriptions for each datum according to the binary tree. This decreases the amount of data contained in the data set.
	Compression Process	Subsequently, apply Huffman coding to get a new data set that is less in size and has subtle patterns hard to be identified.
Data Encryption [34]	Selection of Encryption Algorithm	Select a well tested encryption standard (for instance Advanced Encryption Standard, AES) proportional to the compressed data set.
	Encryption Process	Compress the data and than encrypt the compressed data according to chosen algorithm. The encryption also guarantees that the encoded data is safe from an external attempt either by force or by default.
	Key Management	For encryption keys, it is possible to adopt a key management best practice and provide for proper security of the keys. This system should have ways of creating and distributing them, storing them and regularly changing them to improve security.
Decompression and Decryption [35]	Decryption Process	When the access to the data is needed first, decrypt the compressed data through decryption algorithm to be used.
	Decompression Process	Here, Huffman decoding must be used to try to get the original data back from the decrypted compressed dataset.

The proposed model has several desirable features: increasing the compression and encryption of data and subsequently reducing storage and transmission needs, as well as improving efficiency in the processing. In this way, the model meets the security and performance requirements for knowledge databases in combination with these techniques [36]. The (Table 7) highlights three key system features: There are benefits known as security, efficiency, and resource utilization that have been enhanced. Enhanced security in the database compresses and encrypts the manner in which data patterns are ensured so that any unauthorized persons do not access them. Efficiency improvement is one of the pre-encryption processing methods that helps you preserve time when shrinking the data size that has been observed as time-consuming. Last, optimization of storage and bandwidth is achieved by optimal resource utilization, making it suitable for large datasets in situations where resources are limited. These features make the system secure, scalable, and efficient for arising modern apps.

Table 7. Key Features and Benefits of the Proposed Model

Feature	Description
Enhanced Security	Through combining compression and encryption the model hides data patterns and has a strong defence against the unauthorized access.
Efficiency Improvement	Whenever the data is huge, it is compressed in order to minimize the amount of computation that must be done to encrypt the data and enhance the rate of processing.
Optimal Resource Utilization	The model is efficient for storage requirement and bandwidth hence suitable for large dataset environments.

From the (Table.8), it can be concluded that, the model has significant advantages, given certain limitations. The combined processes of compression with encryption may be a problem to implement because they are more complex and may demand more resources. However, the effectiveness of the model depends on the peculiarity of the data and the knowledge database used [37], [38].

Table 8. Challenges and Solutions in Huffman Coding Implementation

Challenge	Details	Ponder Points
Dynamic Data Management	Specifically, the model may require modification in scenarios where databases have data that changes often. This could be resolved through implementation of adaptive Huffman coding.	Finally, do look into the adaptive Huffman coding to add flexibility in the system.
Overhead of Huffman Tree	The cost of transmitting the Huffman tree can be accommodated by comes in the form of optimally pre-defining Huffman trees for often used datasets.	Fixed Huffman trees should be applied to frequently used data sets.

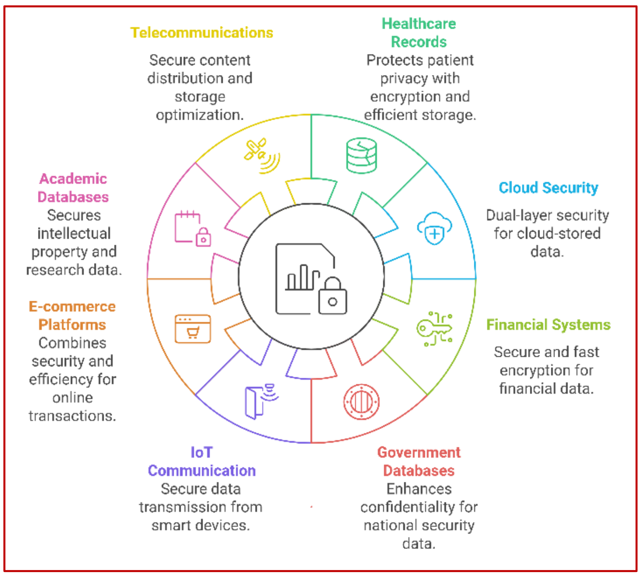

Telecommunications
Secure content distribution and storage optimization.
Healthcare Records
Protects patient privacy with encryption and efficient storage.
Cloud Security
Dual-layer security for cloud-stored data.
Financial Systems
Secure and fast encryption for financial data.
Government Databases
Enhances confidentiality for national security data.
IoT Communication
Secure data transmission from smart devices.
E-commerce Platforms
Combines security and efficiency for online transactions.
Academic Databases
Secures intellectual property and research data.

Figure 9. Real-World Applications of Enhanced Data Security Using Fast Huffman Encoding and Encryption Techniques

Real World Application

The paper depicts (Figure .8) applicability in various fields. In healthcare, it guarantees the safe storage and transfer of highly confidential EHRs besides efficient storage. For cloud computing, it offers security features for data at rest and data in transit. Both the financial and e-commerce sectors benefit because transaction data and customer information are protected better and take up less storage space. This approach is used in government and military systems that provide security to classified information as well as in efficient database systems. In IoT ecosystems, it plays the role of authenticating the parties involved and encrypting and compressing data for transmission. It can be used by the institutions of learning to protect invention and research data sets. It also helps enterprises in the analysis of Big data since it provides secure storage of big data sets. Moreover, it improves blockchain and cryptocurrency systems by safeguarding records of transactions and at the same time, the approach is cost-effective to store.

CONCLUSION

This paper proposes a new model in order to improve data protection in knowledge databases by using data compression via Huffman encoding and then encryption. By comparison, with the proposed model, data security is enhanced while adopting less storage space, less communication overhead, and faster processing time. This leads to the conclusion that additional work will be dedicated to the enhancement of this model and the investigation of other application domains for which this approach can be useful. The fast Huffman encoding architecture described in this paper is developed onto an FPGA, which uses the Xilinx Zynq7035 with a working frequency of 200 MHz. Firstly, the work delves into the main concepts and organization of Huffman encoding with the use of two-speed encoding structures and byte recombination in order to reduce the number of cycles used in processing. As shall be seen from the experimental results, although the method employs some LUTs, it obtains fewer clock cycles than the existing method, thus making real-time encoding in the JPEG compression possible. In this work, an adaptive fast Huffman encoding technique is used. The multi-MSBs of non-reference pixels are predicted by MED and GAP, jointly embedded into a loop. Following this, adaptive Huffman coding is performed on all the bits in the order of the highest to the lowest order for the original as well as the predicted pixel intensities. The image is then encrypted by a stream cipher, and secret data is inserted into the spare area by multi-MSB substitution. New experimental outcomes illustrate that this method—regarding combined MED+GAP predicting and adaptive Huffman coding—provides larger embedding capacity than similar methods and assures the same level of robust security and reversibility. This pioneering approach demonstrates that adaptive encoding schemes, in theory, can be easily optimized for both mass compression and superior safety while achieving high speed.

FUTURE WORK

The proposed research will also design the model for a wide range of knowledge databases to check the model's flexibility in many situations. A qualitative research study will be done with the aim of assessing various factors that comprise the above theoretical model in the study, and these may include the level of compression ratio and the enhanced security achieved through the encryption and decryption exercise. Besides, the thesis will explore the application of other forms of compression to the model with an aim of improving resource utilization and performance on large and complex data sets. This multi-faceted endeavour aims to legitimize the model as a viable, confidentiality-preserving solution for banking data storage.

REFERENCES

[1] S. Lysenko, K. Bobrovnikova, R. Shchuka, and O. Savenko, "A Cyberattacks Detection Technique Based on Evolutionary Algorithms," *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, May 2020, doi: https://doi.org/10.1109/dessert50317.2020.9125016.

[2] D. Liu, P. An, R. Ma, W. Zhan, X. Huang, and Ali Abdullah Yahya, "Content-Based Light Field Image Compression Method With Gaussian Process Regression," *IEEE Transactions on Multimedia*, vol. 22, no. 4, pp. 846–859, Mar. 2020, doi: https://doi.org/10.1109/tmm.2019.2934426.

[3] K. R. Raghunandan, Radhakrishna Dodmane, K. Bhavya, K. Rao, and A. K. Sahu, "Chaotic-Map Based Encryption for 3D Point and 3D Mesh Fog Data in Edge Computing," *IEEE Access*, vol. 11, pp. 3545–3554, Dec. 2022, doi: https://doi.org/10.1109/access.2022.3232461.

[4] C. Bhatt, C. Tiwari, B. S. Thalal, A. Vishnoi, K. Joshi, and T. Singh, "Secured Multi-Platform Communication Application Using Advanced Encryption Standard Algorithm," *2024 10th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp. 483–487, Mar. 2024, doi: https://doi.org/10.1109/icaccs60874.2024.10716832.

[5] B. Kumari, N. K. Kamal, A. M. Sattar, and M. K. Ranjan, "Adaptive Huffman Algorithm for Data Compression Using Text Clustering and Multiple Character Modification," RECENT TRENDS IN PROGRAMMING LANGUAGES, vol. 10, no. 1, pp. 30–40, May 2023, doi: https://doi.org/10.37591/rtpl.v10i1.509.

[6] MVS Srimanth and Ravi Kumar Jatoth, "Implementation Challenges and Performance Analysis of Image Compression Using Huffman Encoding and DCT Algorithm on DSP Processor TMS320C6748 and Arduino Nano 33 BLE," vol. 5, pp. 1–6, Feb. 2024, doi: https://doi.org/10.1109/sceecs61402.2024.10481684.

[7] D. Castro, B. Leite, and Cleber Zanchettin, "An End-to-End Approach for Handwriting Recognition: From Handwritten Text Lines to Complete Pages," pp. 264–273, Jun. 2024, doi: https://doi.org/10.1109/cvprw63382.2024.00031.

[8] J. Tekli, "An Overview on XML Semantic Disambiguation from Unstructured Text to Semi-Structured Data: Background, Applications, and Ongoing Challenges," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1383–1407, Jun. 2016, doi: https://doi.org/10.1109/tkde.2016.2525768.

[9] S. Prasad and Y. Sreenivasa Rao, "Designing Secure Data Storage and Retrieval Scheme in Cloud-Assisted Internet-of-Drones Environment," *IEEE Internet of Things Journal*, pp. 1–1, Jan. 2024, doi: https://doi.org/10.1109/jiot.2023.3337265.

[10] Wahyu Sardjono, Desi Maya Kristin, and Gustian Rama Putra, "Model of Customer Relationship Management Systems Evaluation Using Factor Analysis," Aug. 2023, doi: https://doi.org/10.1109/icimtech59029.2023.10277947.

[11] A. R. Hakim, K. Ramli, T. S. Gunawan, and S. Windarta, "A Novel Digital Forensic Framework for Data Breach Investigation," IEEE Access, vol. 11, pp. 1–1, 2023, doi: https://doi.org/10.1109/access.2023.3270619.

[12] Y.-T. Pai, F.-C. Cheng, S.-P. Lu, and S.-J. Ruan, "Sub-Trees Modification of Huffman Coding for Stuffing Bits Reduction and Efficient NRZI Data Transmission," IEEE Transactions on Broadcasting, vol. 58, no. 2, pp. 221–227, Jun. 2012, doi: https://doi.org/10.1109/tbc.2012.2189610.

[13] I. Timokhin and F. Ivanov, "Fast Polar Decoding With Successive Cancellation List Creeper Algorithm," IEEE Access, vol. 12, pp. 86639–86648, 2024, doi: https://doi.org/10.1109/access.2024.3416826.

[14] Y.-T. Pai, F.-C. Cheng, S.-P. Lu, and S.-J. Ruan, "Sub-Trees Modification of Huffman Coding for Stuffing Bits Reduction and Efficient NRZI Data Transmission," *IEEE Transactions on Broadcasting*, vol. 58, no. 2, pp. 221–227, Jun. 2012, doi: https://doi.org/10.1109/tbc.2012.2189610.

[15] J. Meng, L. Liu, Y. Liu, and N. Wang, "WD: A Sliding Window based Time Series compression algorithm," vol. 17, pp. 148–152, Dec. 2023, doi: https://doi.org/10.1109/mlbdbi60823.2023.10482339.

[16] B. Lian et al., "Trusted Location Sharing on Enhanced Privacy-Protection IoT Without Trusted Center," IEEE Internet of Things Journal, vol. 11, no. 7, pp. 12331–12345, Nov. 2023, doi: https://doi.org/10.1109/jiot.2023.3336337.

[17] S. More, Vrinda Sanivarapu, Y. Sharma, Ved Milind Thigale, and M Suguna, "Enhancing Data Compression: A Dynamic Programming Approach with Huffman Coding and Burrows-Wheeler Transform," vol. 52, pp. 82–88, Dec. 2023, doi: https://doi.org/10.1109/icacrs58579.2023.10404627.

[18] W. Wang and W. Zhang, “Huffman Coding-Based Adaptive Spatial Modulation,” *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 5090–5101, Aug. 2017, doi: https://doi.org/10.1109/twc.2017.2705679.
[19] S. A. Ramprashad, “The multimode transform predictive coding paradigm,” IEEE Transactions on Speech and Audio Processing, vol. 11, no. 2, pp. 117–129, Mar. 2003, doi: https://doi.org/10.1109/tsa.2003.809195.
[20] Eiji Kasutani and A. Yamada, “The MPEG-7 color layout descriptor: a compact image feature description for high-speed image/video segment retrieval,” International Conference on Image Processing, Oct. 2001, doi: https://doi.org/10.1109/icip.2001.959135.
[21] J. Zepeda, C. Guillemot, and E. Kijak, “Image Compression Using Sparse Representations and the Iteration-Tuned and Aligned Dictionary,” vol. 5, no. 5, pp. 1061–1073, Apr. 2011, doi: https://doi.org/10.1109/jstsp.2011.2135332.
[22] T. Kumaki et al., “CAM-based VLSI Architecture for Huffman Coding with Real-time Optimization of the Code Word Table,” pp. 5202–5205, Jul. 2005, doi: https://doi.org/10.1109/iscas.2005.1465807.
[23] A. Gupta, M. C. Srivastava, S. D. Pandey, and V. Bhandari, “Modified Runlength Coding for Improved JPEG Performance,” International Conference on Information and Communication Technology, vol. 23, pp. 235–237, Mar. 2007, doi: https://doi.org/10.1109/icict.2007.375383.
[24] G. Baruffa, Pisana Placidi, A. D. Salvo, S. Marconi, and A. Paterno, “An Improved Algorithm for On-Chip Clustering and Lossless Data Compression of HL-LHC Pixel Hits,” Nov. 2018, doi: https://doi.org/10.1109/nssmic.2018.8824281.
[25] J. Li, G. AlRegib, D. Tian, P. S. Chang, and W. H. Chen, “Three-dimensional position and amplitude VLC coding in H.264/AVC,” 2008 15th IEEE International Conference on Image Processing, pp. 2488–2491, 2008, doi: https://doi.org/10.1109/icip.2008.4712298.
[26] L. S. Ajay and Sreenidhi Prabha Rajeev, “Comparative Analysis of Data Compression using Canonical Huffman and Golomb Rice Encoding in Verilog HDL and Implementation in FPGA,” Jul. 2023, doi: https://doi.org/10.1109/icccnt56998.2023.10306693.
[27] F. Shang, Q. Ding, R. Du, M. Cao, and H. Chen, “Construction and Application of the User Behavior Knowledge Graph in Software Platforms,” Journal of Web Engineering, Mar. 2021, doi: https://doi.org/10.13052/jwe1540-9589.2027.
[28] J. Kang and S.-J. Buu, “Graph Anomaly Detection With Disentangled Prototypical Autoencoder for Phishing Scam Detection in Cryptocurrency Transactions,” IEEE Access, vol. 12, pp. 91075–91088, Jan. 2024, doi: https://doi.org/10.1109/access.2024.3419152.
[29] J. Radhakrishnan, S. Sarayu, K. George Kurian, D. Alluri, and R. Gandhiraj, “Huffman coding and decoding using Android,” Apr. 2016, doi: https://doi.org/10.1109/iccsp.2016.7754156.
[30] Janarbek Matai, J.-Y. Kim, and R. Kastner, “Energy efficient canonical huffman encoding,” Jun. 2014, doi: https://doi.org/10.1109/asap.2014.6868663.
[31] Ramez Moh. Elaskary, M. Saeed, T. Ismail, H. Mostafa, and Salam Gabran, “Hybrid DCT/Quantized Huffman compression for electroencephalography data,” vol. 2, pp. 111–114, Dec. 2017, doi: https://doi.org/10.1109/jec-ecc.2017.8305790.
[32] J. Choi, B. Kim, H. Kim, and H.-J. Lee, “A High-Throughput Hardware Accelerator for Lossless Compression of a DDR4 Command Trace,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 1, pp. 92–102, Jan. 2019, doi: https://doi.org/10.1109/tvlsi.2018.2869663.
[33] Y. Yu, Z. Zhao, S.-J. Lin, and W. Li, “Accelerating Huffman Encoding Using 512-Bit SIMD Instructions,” IEEE Transactions on Consumer Electronics, vol. 70, no. 1, pp. 554–563, Dec. 2023, doi: https://doi.org/10.1109/tce.2023.3347229
[34] F. Shang, Q. Ding, R. Du, M. Cao, and H. Chen, “Construction and Application of the User Behavior Knowledge Graph in Software Platforms,” Journal of Web Engineering, Mar. 2021, doi: https://doi.org/10.13052/jwe1540-9589.2027
[35] J. Haghighat, W. Hamouda, and M. R. Soleymani, “Lossless Source Coding Using Nested Error Correcting Codes,” IEEE Transactions on Signal Processing, vol. 55, no. 6, pp. 2583–2592, Jun. 2007, doi: https://doi.org/10.1109/tsp.2007.893934
[36] D. Puthal, X. Wu, N. Surya, R. Ranjan, and J. Chen, “SEEN: A Selective Encryption Method to Ensure Confidentiality for Big Sensing Data Streams,” IEEE Transactions on Big Data, vol. 5, no. 3, pp. 379–392, Sep. 2019, doi: https://doi.org/10.1109/tbdata.2017.2702172
[37] R. Gupta, “Lossless Compression Technique for Real-Time Photoplethysmographic Measurements,” IEEE transactions on instrumentation and measurement, vol. 64, no. 4, pp. 975–983, Apr. 2015, doi: https://doi.org/10.1109/tim.2014.2362837
[38] D. Yunge, S.-Y. Park, P. H. Kindt, and S. Chakraborty, “Dynamic Alternation of Huffman Codebooks for Sensor Data Compression,” IEEE Embedded Systems Letters, vol. 9, no. 3, pp. 81–84, Sep. 2017, doi: https://doi.org/10.1109/les.2017.2714899