Comparison of CRC16 and PNC16 Models to Identify Errors in Python

Odilzhan Turdiev[1], Masud Masharipov[1], Maad Mudher Khalil[2] and Mohammed Sami Mohammed[2]
[1]*Departments of Information Systems and Technologies in Transport, Tashkent State Transport University, Temiryolchilar Str. 1, 100167 Tashkent, Uzbekistan*
[2]*University of Diyala, 32009 Baqubah, Diyala, Iraq*
odiljan.turdiev@mail.ru, masudcha@mail.ru, maadalomar@gmail.com, dr.mohammed.sami@uodiyala.edu.iq

Keywords: Python, Cyclic Redundancy Code, Probability Code Number, Error Detection.

Abstract: In the modern world of programming, where reliability and security are critically important aspects, the detection and correction of errors becomes an integral part of software development. One of the methods for detecting errors is the use of error codes, such as Cyclic Redundancy Checks (CRC) and probabilistic number code (PNC). In this work, we compare these two models for detecting errors in the python programming language. The aim of the study is to investigate the efficiency and applicability of these models for detecting errors in data, including 6-bit errors. Full-fledged code examples are provided for each model. These models are involved to provide analyzation of its contribution and how it deals with errors, which ensures data integrity through the full process. In addition, the performances of CRC and PNC for 6-bits are included and studied for this purpose. Results showed that CRC16 provide better performances than PNC16. The high reliability of CRC16 is due to restrict mathematical operations that CRC16 followed to detect errors. While PNC16 introduced uncertainty and occasional failures in detecting errors for the same data that has been used with CRC16.

1 INTRODUCTION

The reliability of related codes especially in software development is considered a crucial factor for programming process. Some issues related to code errors which lead to undesired faults such as incorrect behavior or even some notable security vulnerabilities. Different types of models with various tools was applied to specify and try to solve such issues [1], [2]. From these common techniques are Cyclic Redundancy Check (CRC) and Probable Number Code (PNC), which designed for data transmissions. In spite of same field utilization for these both techniques, but different approaches are defined for each technique. Based on Python software, this article provide a comparison between CRC and PNC according to error samples provided by users. Using Python provided an exploration about each technique properties like strengths in addition to their weaknesses points. The goal was also for providing some recommendations on how the choice would be to specify the appropriate method for a specific task. According to the software performances, programmer or even users can examine these techniques for better decision reports. Also, system needs with its related constraints specify which one of these techniques are more suitable for a specific task.

The main goal of this article is the using of direct utilization of python software to give the comparison between these two techniques. These would provide more points about these techniques which cannot be observed in theoretical rules.

2 ANALYZING THE CRC ALGORITHM

Different error types could be occurred due to the interferences in the form of transmitted frame, these types are defined as single, multiple or in a packet form.

Packet error is defined as the number of bits between two consecutive erroneous bits Figure 1. In addition, when determining the packet error length, the last erroneous bit in a packet and the first erroneous bit in the next packet must be separated [1].

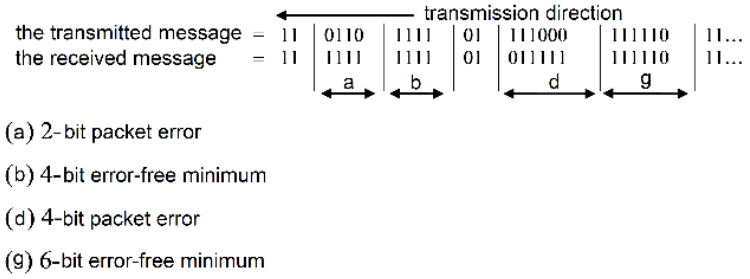

Figure 1: Example of a packet error.

The CRC check summing method is based on the properties of division with remainder of a polynomial (binary number). In essence, the CRC result is the remainder from dividing the polynomial corresponding to the original data by the generating polynomial of fixed length.

The standard way of representing a generating polynomial is to show those positions at which the binary units are powers of X. Examples of generating polynomials used in practice are as follows [2]-[3]:

$$CRC16 = x^{16} + x^{15} + x^2 + 1;$$

$$CRC - CCITT = x^{16} + x^{15} + x^5 + 1.$$

Hence, CRC-16 in binary form is equivalent to writing:

11000000000000101.

With this generator polynomial, 16 zeros will be added before the generation of the FCS (Frame Check Sequence). The last one will be a 16-bit remainder.

CRC-16 and CRC-CCITT are widely used in networks such as ISDN, while CRC-32 is used in most local area networks. The CRC method can be easily implemented in hardware and software.

One set of check digits is generated (calculated) for each transmitted frame based on the frame contents and added by the transmitter to the tail of the frame. The receiver then performs a similar calculation on the full frame plus the check digits. If no errors were found, there should always be a known result; if a different answer is received, this indicates an error.

The number of check digits per frame is chosen according to the expected type of transmission errors; 16 and 32 bits are the most common. The calculated check digits are labeled as FCS or cyclic redundancy CRC frame check sequence [4].

Essentially, the method utilizes a property of binary numbers. When using modulo 2 arithmetic [5, 6]:

M(x) –k -digit -number (the message to be transmitted);

$G(x)$ – $(n+1)$ -bit number (divisor or generator);

$R(x)$ – n is a digit -number such that k> n (remainder);

$\frac{M(x)*2^n}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$, where Q(x) is private;

$\frac{M(x)*2^n + R(x)}{G(x)} = Q(x)$, assuming arithmetic modulo2.

This result can be easily confirmed by substituting the expression for M(x) * 2^n /G(x), into the second equation:

$$\frac{M(x)*2^n + R(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)} + \frac{R(x)}{G(x)},$$

equal to Q(x), since all numbers modulo 2 added to it will be equal to zero, i.e. the remainder will be equal to zero.

To use the full frame contents M(x) along with the added set of zeros equal to the number of FCSs to be generated (i.e., multiplied by 2^n, where n is the number of FCSs) is divided modulo 2 by a binary number G(x) is the generator polynomial containing one more unit than FCSs). The division operation is equivalent to performing an exclusive OR operation on a parallel bit, since every bit in the frame is processed. Then the remainder of R(x) is FCS, which is transmitted at the tail of the information frames.

Similarly, when received, the received bit stream, which includes the CRC number, is again divided by the same generator polynomial, i.e.

$M(x) * 2^n + R(x)/G(x)$, and if there is no error, the remainder is all zeros. However, if an error is present, the remainder is not zero.

The choice of the generating polynomial is important because it determines the types of errors detected. Suppose that the transmitted frame:

$$M(x) = 110101100110,$$

and the error pattern

$$E(x) = 000000001001.$$

Thus, 1 in the bit position indicates an error. Let's apply the Boolean function sum modulo 2.

The resulting frame = M(x) + E(x)

$$\frac{M(x)+E(x)}{G(x)} = \frac{M(x)}{G(x)} + \frac{E(x)}{G(x)}$$

Since M(x) /E(x) does not give a remainder, the error is present if E(x)/ G(x) gives a remainder.

For example, G(x) has at least three non-zero summands (1 bits) and E(x)/ G(x) will give a remainder for all one-bit and all two-bit errors with modulo 2 arithmetic, and hence the errors are detectable. Conversely, an error of length(x) gives no remainder and goes undetected [2].

The generator polynomial of *R* bits detects [7]:

- all one-bit errors,
- all two-bit errors,
- all odd numbers of bit-errors,
- all error packets< R,
- most error packets> R.

Cyclic redundant CRC codes are a subclass of block codes and are used in HDLC, Token Ring, TokenBus, Ethernet protocol families and other link layer protocols [14]. Computational resources are understood as memory, processor power, and the number of shift registers [12]. One of the ways to represent a cyclic code is to represent it as a generating polynomial - a set of all polynomials of degree (r-code-1) containing as a common multiplier some fixed polynomial $G(x)$. The polynomial $G(x)$ is called the generating polynomial of the code. For example, $x^4 + x + 1$, here r-code = 5, since the binary sequence looks like 10011. The standardized and recommended generating polynomials for the CRC algorithm are which shows the name of the standard and the generating polynomial [8]: for example, the entry $x^4 + x + 1$ is equivalent (in binary) to

$$1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 1 \cdot x^0 = 10011 .$$

2.1 Algorithms for Calculating CRC16

CRC16 (Cyclic Redundancy Code) is a data integrity method that uses a polynomial of degree 16 to calculate a checksum [13].

The algorithm for calculating CRC16 is as follows:

1) Initialization: Set the initial CRC16 value to 0xFFFF.
2) For each byte of data:
   - Invert the bits of a data byte.
   - Add a data byte to the current CRC16 value.
   - For each bit of data byte:
   - If the XOR between the current CRC16 value and the current bit is 1, perform a right shift operation by 1.
   - Otherwise, perform a right shift operation by 1 without changing CRC16.
3) The final value of CRC16 is a checksum, which is a 16-bit value.

Example:
For data 0x21 0x43 0x65, calculate CRC16.
Initial CRC16 value: 0xFFFF.

Step 1. For byte 0x21:
- Inverted byte: 0xDE.
- CRC16 = 0xFFFF + 0xDE = 0xF4DD.

Step 2. For byte 0x43:
- Inverted byte: 0xBC.
- CRC16 = 0xF4DD + 0xBC = 0xF579.
- Check each bit and make the appropriate shifts.

Step 3. For byte 0x65:
- Inverted byte: 0x9A.
- CRC16 = 0xF579 + 0x9A = 0xFF13.
- Check each bit and make the appropriate shifts.

The final CRC16 value for data 0x21 0x43 0x65 is 0xFF13.

2.2 Strengths and Weaknesses Points of the CRC16 Model

CRC16 like other error detection techniques has privilege points which may be summarized as followed:

1) CRC16 is built in easy and efficient way for different devices or platforms.
2) Due to its quick calculations process, CRC16 is suited for plenty type of applications.
3) CRC16 has effective error performances in error identifying especially for transmission data.

In the other hand, CRC16 has some issues which limit their usage which could be mentioned as follow:

1) Errors detection need correction process which cannot be provided by CRC16. This issue makes the system to retransmit data to the sender side and taking too long time with cost maximization.
2) Dealing with larger data makes the working of CRC16 limited which provide restriction in detection process.
3) calibration between parameters and initial values may take too much time to be carefully selected and finalized, which need additional efforts.

3 ANALYSIS OF THE PNC16 ALGORITHM

This technique is considered as mathematical process which utilize to specify the likelihood of selection approach. These numbers are randomly selected and contained a specific number of digits. Some applications needed a probability estimation could use of PNC for their evaluations. Data transmission checking from errors and storage field are such an applications need PNC for numerical events estimation process.

The encoded operation for a set of data could be corrupted with faults during transmission process. These changing in data presentation need an error detection using specific technique based on their tasks.

The PNC estimate the probability of corrupted data for data transmission systems which is affected by interferences. The decision of this technique is also effected by the data accuracy, the PNC is also prompt any additional verifications. As a result, the PNC is considering as an impact tool to improve the transmission process with error detection and providing an information acknowledgment about data likelihood.

The probability of selecting a number for one digits would be 0.1 which ranging from (0 to 9) for a single digit that has been randomly selected. These values are independent of other digits values that could be appearing in any given positions of related or transmitted data. The certain numbers of combination probability appearing could be calculated using equation below:

$$PNC = (Nk) / 10n$$

where PNC is number of coding probability:

- N is the possible appearing digits (which is equal to 10 for a regular system)
- k specifies number of digits that required for a specific combination.
- n is the total number of digits in the number.

3.1 Algorithm for Calculating PNC16

Creation of high quality, fast and simple enough algorithms of formation of checksums with the help of parallel random number generator is one of the main problems of data transmission with the help of low-frequency energy-saving systems. The solution of this problem ultimately determines the success of building a PNC model, since the characteristics of the parallel random number generator (PRNG) largely determine the parameters of the PNC [9].

The relevance of the issue of synthesis of the model of formation of the probable code of PNC number is closely connected with the relevance of the problem of implementation of the principles of probabilistic methods of modeling and calculations of checksums of data transmitted over communication channels [10], [11].

Figure 2 presents the general scheme of the model of formation of the probable code of PNC number, where A - transmitted binary sequence of digit n, Xi - formed by the i-th parallel PRNG $\left(i \in \{1,...,M\}\right)$ random code of digit n, Ki - result of bitwise logical operation over the transmitted binary sequence and random code of digit n, si - i-th element of the calculated checksum, S - M-bit checksum (probable number code) transmitted together with the binary sequence over communication channels.

Checksum calculation for a binary sequence

$$A = (a_1, a_2, a_3, ..., a_n), \; a_i \in \{0;1\}$$

is performed by means of several parallel pseudorandom number generators (PRNGs). The i-th $\left(i \in \{1,...,M\}\right)$ PRNG generates the random sequence

$$X_i = (x_{i,1}, x_{i,2}, x_{i,3}, ..., x_{i,n}), \; x_{i,j} \in \{0;1\}, \; j \in \{1,...,n\},$$

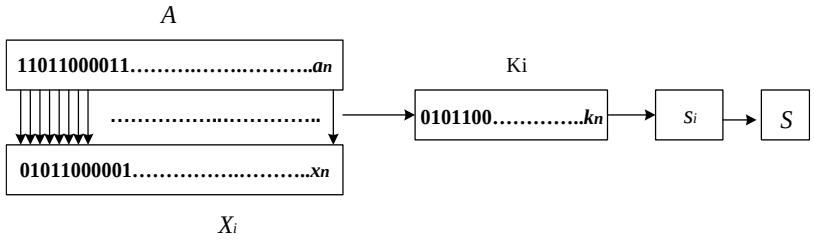

Figure 2: General scheme of the PNC formation model.

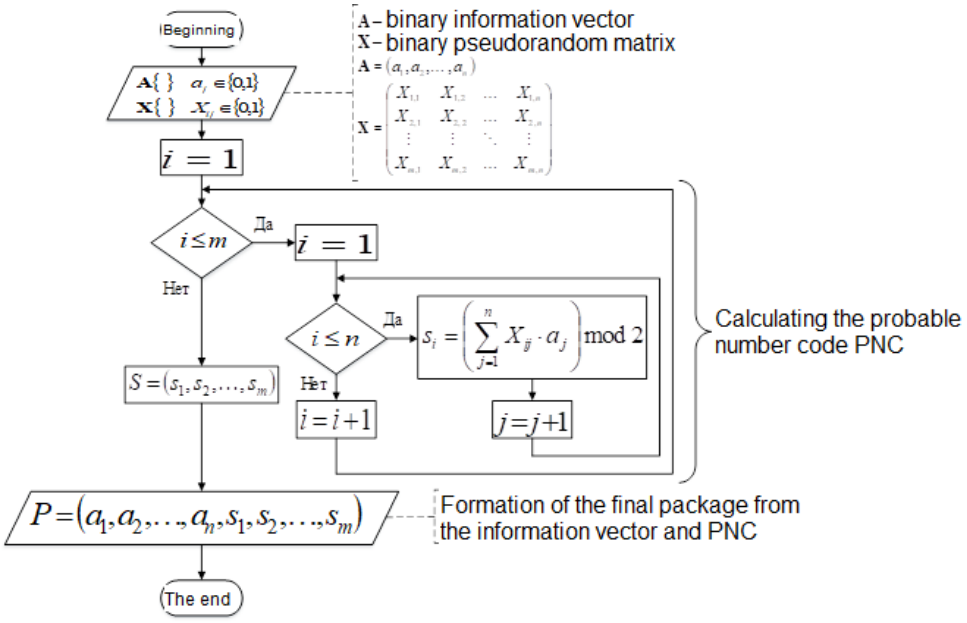

Figure 3: Block diagram of the algorithm of PNC checksum formation.

where by elements of the sequences with the elements of sequence A are subjected to the logical operation AND (denoted by the symbol &) followed by summation by mod2. As a result, the following binary sequences are obtained:

$$K_i = A \& X_i = (k_{i,1}, k_{i,2}, k_{i,3}, \ldots, k_{i,n}),$$

where $k_{i,j} = a_j \& x_{i,j}$.

The checksum element is obtained by:

$$s_i = \left(\sum_{j=1}^{n} k_{i,j} \right) \bmod 2 .$$

Thus, the checksum is a binary sequence

$$S = (s_1, s_2, s_3, \ldots, s_M),$$

where *M* is the number of CBCPs used.

All these activities can be represented in matrix form as follows. The elements form a matrix:

$$\mathrm{x} = \begin{pmatrix} \mathrm{x}_{1,1} & \mathrm{x}_{1,2} & \cdots & \mathrm{x}_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{pmatrix}.$$

Here - is the matrix of the HGPSP matrix, - is a random uniformly distributed integer taking values from the set However, the first row of this matrix consists of units only. Elements form a vector

$$A = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}.$$

On the transmitter line, the sum is calculated as follows: the elements of matrix X are line by line multiplied by the elements of vector A (source code),

the result is the sum of each row sm, and matrix S represents the checksum (PNC).

$$S = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_m \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{pmatrix}.$$

If the transmissions and S received interference and failures, we will get $\tilde{A}$ and $\tilde{S}$ in the receiver ,

$$S = \tilde{A} * X.$$

To detect interference and failures, we need to compare the check code with $\tilde{S}$, if $S \neq \tilde{S}$ is not equal - there is an error, if it is equal - there is no error.

Figure 3. shows a block diagram of the PNC checksum generation algorithm.

Example: A = 11010, with a 5-digit PNC Figure 4.

Calculation result: A+S = 1101010000

$$\begin{matrix} S & & A & & & X & & & & \end{matrix}$$
$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Figure.4. Example of PNC residue calculation.

Also, this operation can be represented by the following formula:

$$S_i = X_{ij} \cdot A_i.$$

An example with probable interference is shown in Figure 5.

Received packet with error: 01010 10100

Reverse calculation result: 01010 00111

$$\begin{matrix} \widetilde{S} & & \widetilde{A} & & & X & & & & \end{matrix}$$
$$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Figure.5. Example of PNC calculation with errors (interference).

Based on the analysis of the above-mentioned material, and in order to verify the error-free model, the author considers different ways of solving the problem in the next chapter within the framework of developing a method of data integrity control based on stochastic calculations.

3.2 Advantages and Disadvantages of the PNC16 Model

Advantages of the PNC16 model:

1) The PNC16 is easy and intuitive to use, making it accessible to a wide range of users.
2) CRC16 allows you to quickly and accurately estimate the probability of a number code, which helps you make more informed decisions.
3) Based on special algorithms and models, the PNC16 model provides high accuracy number code probability calculations.

Some issues that PNC are suffering from or consider as disadvantages as followed:

1) PNC16 has limitations such as storage requirements, non-deterministic behavior in addition to error high sensitivity [15].
2) PNC16 also need additional parameters like synchronization parameters, threshold value and initial population values. Which mean more computational evaluations to provide accurate calculations to a set of number code.
3) It directly depends on data quality, which make PNC16 affected by noise and lead to distort the probability distribution, resulting in misclassification or incorrect predictions. It is also affected by missing values that could lead to reduce reliability.

4 CONCLUSIONS

In this article, two models have been utilized (CRC16 and PNC16) to provide analysis of 6-bit error data using Python. CRC16 was applied by using the checksum value that was extracted from the data bits themselves and comparing it with the desired value. When the checksum is not equal, then an error has occurred. In PNC16, it relies on a probabilistic number code and compares the threshold value with the count of a set of bits. Through the comparison process, CRC16 provides better performance than PNC16. The high reliability of CRC16 is due to the strict mathematical operations that CRC16 follows to detect errors, while PNC16 introduces uncertainty and occasional failures in detecting errors for the same data handled by CRC16. On the other hand, CRC16 is less sensitive to the input dataset than PNC16 due to the wide range of polynomial-designed functions. Also, PNC16 is more affected by noise and interference that could occur in the channel, leading to misclassification in error

detection. CRC16 also, in general, generates a fixed-length checksum value which allows for direct error processing with different types such as single, burst, and multiple errors. In PNC16, it depends on probabilistic analysis regardless of checksum values, which fails in detecting errors and is specified for a single error type. CRC16 provides better results than PNC16 in error detection due to its deterministic nature, strong polynomial theory, and reduced dependency on the data itself. However, PNC16 may miss errors due to being affected by data noise and having weaker mathematical guarantees for error detection.

REFERENCES

[1] O.A. Turdiev, "A model for the formation of a probable number code based on stochastic calculations," Intelligent Technologies in Transport, no. 4, pp. 28-33, 2021.

[2] A.H. Saleh and M.S. Mohammed, "Enhancing Data Security through Hybrid Error Detection: Combining Cyclic Redundancy Check (CRC) and Checksum Techniques," International Journal of Electrical Engineering Research, Aug. 2024, [Online]. Available: https://doi.org/10.37391/IJEER.120312.

[3] A. Kotade and A. Nandgaonkar, "Cyclic Redundancy Check: A Novel Software Implementation for machine cycle optimization and analysis for deciding Low Peak to Average Power Ratio Discrete Sequences," in ICCASP/ICMMD-2016, Advances in Intelligent Systems Research, vol. 137, pp. 441-449, 2017.

[4] P.B. Viegas, A.G. de Castro, A.F. Lorenzon, F.D. Rossi, and M.C. Luizelli, "The Actual Cost of Programmable SmartNICs: Diving into the Existing Limits," in Advanced Information Networking and Applications: Proceedings of the 35th International Conference on Advanced Information Networking and Applications (AINA-2021), vol. 1, Springer International Publishing, Germany, 2021, pp. 181-194.

[5] R.N. Williams, Elementary Guide to CRC Algorithms of Error Detection, Aug. 19, 1993.

[6] S.S. Gorshe, "CRC-16 polynomials optimized for applications using self-synchronous scramblers," in Proc. IEEE International Conference on Communications, vol. 5, pp. 2791-2795, 2002, [Online]. Available: https://doi.org/10.1109/ICC.2002.997351.

[7] P. Pramod, R. Anantharaman, and A. Kotain, "FPGA implementation of single bit error correction using CRC," International Journal of Computer Applications, vol. 52, pp. 15-19, 2012, [Online]. Available: https://doi.org/10.5120/8238-1471.

[8] B. Kirocuhenassamy, A. Yessad, S. Jolivet, and V. Luengo, "Toward diagnosis of semantic errors in Python programming platforms for beginners," in Workshop RKDE - ECML PKDD Conference, Vilnius, Lithuania, 2025.

[9] H. Campbell, A. Hindle, and J. Amaral, "Error location in Python: where the mutants hide," 2015, [Online]. Available: https://doi.org/10.7287/peerj.preprints.1132v1.

[10] D. Robinson, N. Ernst, E. Vargas, and M.-A. Storey, "Error Identification Strategies for Python Jupyter Notebooks," 2022, [Online]. Available: https://doi.org/10.48550/arXiv.2203.16653.

[11] M. Arutunian, S. Sargsyan, M. Mehrabyan, L. Bareghamyan, and H. Aslanyan, "Automatic Recognition and Replacement of Cyclic Redundancy Checks for Program Optimization," IEEE Access, pp. 1-1, 2024, [Online]. Available: https://doi.org/10.1109/ACCESS.2024.3518953.

[12] X. Dong and Y. He, "CRC Algorithm for Embedded System Based on Table Lookup Method," Microprocessors and Microsystems, vol. 74, p. 103049, 2020, [Online]. Available: https://doi.org/10.1016/j.micpro.2020.103049.

[13] J. Ray, "Review of Understanding Checksums and Cyclic Redundancy Checks —Philip Koopman (Boca Raton, FL, USA: CRC Press, 2024)," IEEE Reliability Magazine, pp. 1-2, 2024, [Online]. Available: https://doi.org/10.1109/MRL.2024.3389635.

[14] A. Akagic and H. Amano, "A study of adaptable co-processors for Cyclic Redundancy Check on an FPGA," in Proc. International Conference on Field-Programmable Technology (FPT), pp. 119-124, 2012, [Online]. Available: https://doi.org/10.1109/FPT.2012.6412122.

[15] J. Wan, B. Chen, and S. Wang, Smart Manufacturing Factory: Artificial-Intelligence-Driven Customized Manufacturing, CRC Press, 2023.