Rescaling of Symbol Counts for Adaptive rANS Coding

Tilo Strutz
Coburg University, Department of Electrical Engineering and Computer Science
Friedrich-Streib-Str. 2, 96450 Coburg, Germany

***Abstract*—The abbreviation rANS stands for a relatively new method of arithmetic coding based on asymmetric numeral systems (ANS) which combines the advantages of arithmetic coding in terms of performance and the advantages of Huffman coding in terms of speed.**

Compared to conventional arithmetic coding methods, the mathematical apparatus is slightly different which has the consequence that the decoding order is reversed to the encoding order, i.e. the processing follows the last-in-first-out principle. This makes it somewhat difficult to design the coding process to adapt to changing symbol statistics, and therefore rANS coding has so far only been applied in settings with fixed statistics. In particular, the frequent rescaling of statistics required to reduce the influence of old symbols becomes a problem when the order of processing is different on the encoder and decoder sides.

This paper proposes a new method that allows adaptive coding within the framework of rANS coding and additionally offers the possibility of rescaling the symbols frequencies. Investigations show that this method enables the same compression performance for rANS as for conventional arithmetic coding.

***Index Terms*—ANS, adaptive coding, rescaling of statistics, asymmetric numeral system**

I. Introduction

Entropy coding is a fundamental processing step in any efficient data compression system. Based on the given or estimated probabilities p_i of the symbols s_i to be transmitted, the entropy coding step assigns a certain number or fraction of bits to each symbol according to the information content $I(s_i) = \log_2(1/p_i)$. The best known method is Huffman-Coding named after its inventor [1]. It replaces each symbol with the corresponding binary code word. The assignment of bit fraction is not possible with Huffman-Coding. This disadvantage does not apply if arithmetic coding is used instead [2]. Arithmetic coding uses a number interval that is divided into subintervals. The relative width of these subintervals corresponds to the symbol probabilities.

This paper focuses on aspects of a special version of arithmetic coding, the so-called rANS coding [3], [4]. The great advantage of rANS coding is that the required computational steps can be realized by very simple operations if certain parameters are chosen appropriately, allowing a high data throughput (as with Huffman coding) is possible. Moreover, it has the property of achieving a similar reduction of the coding redundancy $\Delta R(x)$ as conventional arithmetic coding. This makes rANS an attractive alternative for applications where Huffman coding has been used so far and a fully-adaptive approach does not promise any advantage.

Due to these benefits, ANS-based coding is already being integrated into commercial systems [5] and has been considered for a new standard in image compression [6].

With conventional arithmetic coding, the decoder may have to read in additional dummy bits to determine the final interval. An rANS decoder, on the other hand, processes only the bits that are actually output by the encoder. This advantage supports the interleaving of two or more encoding processes [7]. In [8] it is studied how real symbol distributions are approximated in ANS systems and [9], [10] investigate the tabular variant (tANS). Tabular ANS coding was also used for image compression in [11]. Index compression for databases is dealt with in [12] and [13]. [14] describes an application of rANS to the compression of detector data recorded during collisions in a particle accelerator. In application to image data compression, rANS is used as an entropy coding technique by [15]. There are also two reviews on ANS, most of which refer to the original publications of Duda [16], [17].

The main disadvantage of rANS coding is that the symbols are processed in last in, first out (LIFO) order making the adaptive processing difficult. So far, it has not been scientifically investigated to what extend the advantages of rANS coding can also be applied to fully-adaptive coding. Its use has until now been limited to scenarios with static, i.e., one-time fixed distribution models. Therefore, rANS coding has not yet been universally applicable and many applications could not benefit from its advantages.

This paper proposes a method that not only facilitates fully-adaptive processing of rANS coding, but additionally provides a technique for rescaling the symbol counts. This rescaling supports faster adaptation by a process of forgetting symbols which have not been observed in the recent past of the coding procedure.

II. Basics

Conventional arithmetic coding

The coding procedure relies on an interval $[0;1)$ that is divided into K subintervals, where K is the number of different symbols (size of the alphabet). The size of the individual subintervals corresponds to the symbols' probability. The individual interval boundaries thus result from the cumulative probabilities:

$$p_\mathrm{k}(s_i) = p_\mathrm{k}(s_{i-1}) + p(s_i) \quad \forall i = 1,2,\ldots,K \qquad (1)$$
$$\text{with} \quad p_\mathrm{k}(s_0) = 0.0\,.$$

Depending on the transmitted symbol, the algorithm zooms into the corresponding subinterval and subdivides it again. With each new symbol to be encoded, the actual subinterval becomes narrower and the current position is represented by a number $0 \le S_n < 1$. With an assumed coding sequence $a, b, c, \ldots$ this number is calculated as follows:

$$S_n := p_\mathrm{k}(a^-) + p(a)\cdot \quad \ldots \\ \Big(p_\mathrm{k}(b^-) + p(b)\cdot\Big(p_\mathrm{k}(c^-) + p(c)\cdot(\ldots)\Big)\Big)\,. \qquad (2)$$

The minus symbol indicates here the lower limit of the subinterval. For example, for the symbol 'a' $p_\mathrm{k}(a^-) = 0$, for 'b' $p_\mathrm{k}(b^-) = p_\mathrm{k}(a) = p(a)$, and for 'c' $p_\mathrm{k}(c^-) = p_\mathrm{k}(b) = p(a) + p(b)$. From (2) it can be seen that the approximate boundaries of the final interval are determined by the first

coded symbol. The further sequence of coded symbols merely refines these limits.

For a practical implementation, all interval boundaries must be scaled to integer values [18], [19]. Absolute frequencies $H(s)$ replace the probabilities:

$$H(s) \approx p(s) \cdot N$$
$$H_{\mathrm{k}}(s) \approx p_{\mathrm{k}}(s) \cdot N \ ,$$

with N as the total number of symbols in the sequence to be encoded. The boundaries of the subintervals are now represented by cumulative frequencies $H_{\mathrm{k}}(s_i)$ instead of cumulative probabilities. The position in the actual interval is determined by the lower interval limit low and the interval width $range$. Progressive interval narrowing is counteracted by regular doubling of the numbers low and $range$. The decoder is able to trace the current interval position back to the transmitted symbol and decodes the symbols in the same order in which they have been encoded: first in, first out (FIFO).

Compared to Huffman coding, arithmetic coding enables significantly better compression, especially for small alphabets and special distribution models. A disadvantage is the higher computational effort and thus time requirement.

There are three possibilities for the operation of an arithmetic coder:

1) Non-adaptive: The occurrence probabilities of the symbols are determined once using a training data set and all cumulative frequencies are determined on this basis. All symbol sequences are processed with this fixed distribution model.
2) Semi-adaptive (static): The symbols of the actual sequence to be coded are counted, all cumulative frequencies are determined on this basis and the sequence is processed with this distribution model.
3) Fully-adaptive (dynamic): The distribution model typically starts with a uniform distribution ($p_i = p$), i.e., all counters are reset at the beginning. After the transfer of a symbol, its counter is incremented and all affected cumulative frequencies are updated.

If the distribution of symbols changes within a sequence to be coded, the fully-adaptive mode of operation can lead to a coding gain because the distribution model automatically adapts to the specific conditions and the transmission of symbols frequencies is not required.

Coding based on asymmetric numeral systems

Asymmetric numeral systems (ANS) offer a new approach to entropy coding [3], [4]. The so-called *range*-ANS coding (rANS) is similar to arithmetic coding in many aspects.

In analogy to the equation (2), the status of the coding process is given by

$$S_n := S_{n-1}/p(s_n) + p_{\mathrm{k}}(s_n^-) \ . \tag{3}$$

Compared to the interval modification according to (2), there are two essential changes. First, the division (instead of multiplication) by the probability $p(s_n)$ of the symbol s_n to be encoded increases the subintervals instead of decreasing them. And secondly, the subsequent addition of the cumulative probability $p_{\mathrm{k}}(s_n^-)$ causes the boundaries of the final interval to no longer depend on the first, but on the last encoded symbol. The latter has drastic effects on the order in which the symbols are processed. The decoder must start with the last encoded symbol and decodes the entire sequence backwards. The rANS coding works according to the LIFO principle: last in, first out.

As with conventional arithmetic coding, the practical implementation requires integers. Let $H(s)$ be the frequency of symbol s and $H_{\mathrm{k}}(s)$ the corresponding cumulative frequency. With the substitutions

$$p(s) = \frac{H(s)}{M} \quad \text{and} \quad p_{\mathrm{k}}(s) = \frac{H_{\mathrm{k}}(s)}{M} \quad \text{with} \quad M = \sum_{\forall s} H(s) \tag{4}$$

we derive from (3):

$$\begin{aligned} S_n &:= S_{n-1} \cdot M/H(s_n) + H_{\mathrm{k}}(s_n^-)/M \\ S_n \cdot M &:= S_{n-1} \cdot M \cdot M/H(s_n) + H_{\mathrm{k}}(s_n^-) \\ \mathbf{S}_n &:= \mathbf{S}_{n-1} \cdot M/H(s_n) + H_{\mathrm{k}}(s_n^-) \ . \end{aligned} \tag{5}$$

As the subintervals (strictly speaking $\mathbf{S}_n$) increase in the process of encoding, the numerical value of $\mathbf{S}_n$ is scaled down when it exceeds a certain threshold.

A fully-adaptive mode of operation, which merely increments the symbol counters as described above and adjusts the symbol distribution in the form of the accumulative frequencies, is not readily possible. Therefore, rANS coding has so far been explicitly recommended as an alternative to Huffman coding with static mode of operation [4].

III. Method

The principles of adaptation and rescaling are first explained for conventional compression systems, in which the encoder and decoder process all symbols in the same order first in, first-out (FIFO). A new method is then proposed for how this can also be achieved for rANS coding, where the decoder has to process the symbols in reverse order, i.e. last in, first out (LIFO).

A. Principle of adaptation and rescaling in FIFO systems

As described in the preceding section, the full-adaptive mode typically starts with equal probabilities for all symbols. All counts are initially set to one. This is the lowest possible count and ensures that the symbols are distinguishable during the coding process. After coding a particular symbol, the encoder and decoder synchronously update the symbol distribution by incrementing the count of the transmitted symbol. In many applications, a drift from one subset of dominant symbols to another subset can be observed. If one wishes to reflect the dominance of the new subset as fast as possible, a mechanism must be introduced that allows to forget the influence of "old" symbols. This can easily be achieved, for example, by scaling down all counts either periodically or at certain events. **Tab. I** shows a toy example with an alphabet $A = \{a, b\}$. Column "s_i" shows the order in which 16 symbols are transmitted. The four columns headed "no rescaling" show how the counts increase, the corresponding probability of the encoded symbol, and the associated information content $I(s_i) = \log_2(1/p(s_i))$ in bit. The probabilities are estimated as the quotient of the individual count of the symbol to be transmitted and the sum of all counts. After the symbol has been sent or has been received by the decoder, its count can be incremented. An ideal entropy coding stage would output a total of 17.05 bit.

The right part of the same table shows how things change when rescaling is done after every four symbols by simply bit-shifting the counts to the right. The first rescaling event

TABLE I
EXAMPLE PROCESS OF ADAPTATION AND RESCALING FOR FIFO SYSTEMS

		no rescaling				with rescaling			
		counts			[bit]	counts			[bit]
i	s_i	a	b	$p(s_i)$	$I(s_i)$	a	b	$p(s_i)$	$I(s_i)$
0		1	1			1	1		
1	a	2		1/2	1.000	2		1/2	1.000
2	a	3		2/3	0.585	3		2/3	0.585
3	a	4		3/4	0.415	4		3/4	0.415
4	a	5		4/5	0.322	5		4/5	0.322
						2	1	rescaling	
5	a	6		5/6	0.263	3		2/3	0.585
6	b		2	1/7	2.807		2	1/4	2.000
7	b		3	2/8	2.000		3	2/5	1.322
8	a	7		6/9	0.585	4		3/6	1.000
						2	1	rescaling	
9	b		4	3/10	1.737		2	1/3	1.585
10	b		5	4/11	1.459		3	2/4	1.000
11	b		6	5/12	1.263		4	3/5	0.737
12	b		7	6/13	1.115		5	4/6	0.585
						1	2	rescaling	
13	b		8	7/14	1.000		3	2/3	0.585
14	b		9	8/15	0.907		4	3/4	0.415
15	b		10	9/16	0.830		5	4/5	0.322
16	b		11	10/17	0.766		6	5/6	0.263
				total:	17.05			total:	12.72

changes the counts from $(5, 1)$ to $(2, 1)$, the second from $(4, 3)$ to $(2, 1)$, and the rescaling after the 12th symbol reduces the counts from $(2, 5)$ to $(1, 2)$. The change from the initially dominant a to the symbol b is now reflected more quickly by the probabilities and the total amount of information to be transmitted is reduced to 12.72 bit. The questions of how often rescaling should be done and how the rescaling is actually carried out need to be answered depending on the specific application and are not discussed in this paper.

B. Principle of adaptation in LIFO systems

The proposed concept for LIFO Systems is not to reverse the symbol order at the decoder stage, but to reverse the order at the encoder. That is, the decoder operates in the same manner as described in the previous subsection. However, the encoder must instead start at index 16 and with the symbol counts that would have been reached after encoding all symbols in forward mode. According to Table I, the counts have to be initialised with 7 for a and 11 for b. This requires the encoder to count all symbols in advance. This is not a disadvantage, since the encoder also has to determine the symbols distribution in semi-adaptive mode in advance and additionally pass this information on to the decoder as side information. During the encoding process, a counter is decremented before the corresponding symbol is encoded.

For better understanding, the pseudo code of encoder and decoder is given in **Listing 1**. The source-code snipped is based on the knowledge of the alphabet size K and the total number N of symbols to be transmitted. The encoder processes all N symbols stored in the array `sequ[]`, the decoder needs this array to save the decoded symbols. A second array `count[]` of size K contains the symbols histogram or distribution. It is initialized by a flat distribution (lines 6-9). The encoder first has to simulate the decoder process (forward processing) in terms of counting the symbol (lines 13-17), because it must start the backward processing with counts that the decoder knows after it has received all N symbols. Then, the encoder processes the symbols of sequence `sequ[]` starting with the last symbol (lines 18-23). Before this symbol can be encoded, its count must be decremented. This procedure is reversing of what the decoder has to do, namely first decoding the symbol, then incrementing the corresponding count and finally storing the symbol in the array (lines 27-32).

Listing 1. Pseudo code of adaptive LIFO coding. See text for details

```
1  N := number of symbols in sequence
2  K := number of different symbols in alphabet
3  sequ[] ... symbol sequence comprising N symbols
4  count[] ... histogram of symbols, K elements

6  for symbol = 0:K-1
7  { // operate as encoder
8    count[symbol] := 1; // initialize array
9  }

11 if encoder_flag
12 {
13   for i = 0:N-1
14   { // count symbols, simulate decoder process
15     symbol := sequ[i];
16     count[symbol] := count[symbol] + 1;
17   }
18   for i = N-1:-1:0
19   { // encode symbols in reverse order
20     symbol := sequ[i];  // get current symbol
21     count[symbol] := count[symbol] - 1; // decrement
           corresponding count
22     encode( symbol, count);// decode symbol
23   }
24 }
25 else // do the decoding
26 {
27   for i = 0:N-1
28   { // decode symbols in normal order
29     symbol := decode( count); // decode symbol
30     sequ[i] := symbol;   // put current symbol
31     count[symbol] := count[symbol] + 1; // increment
           corresponding count
32   }
33 }
```

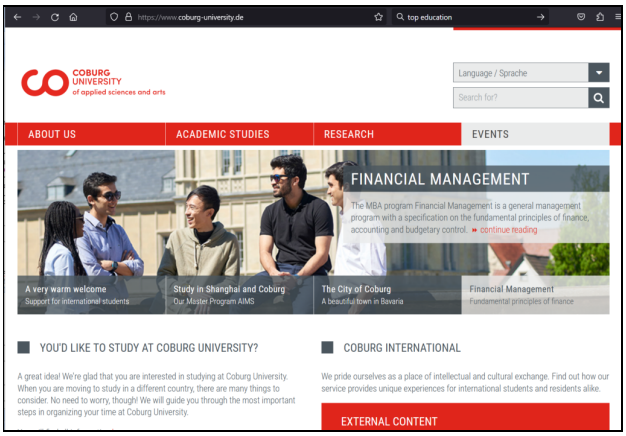
C. Histogram rescaling in LIFO systems

Similar to explanation above, the mode of operation of the rescaling process can also be deduced. The right part of Table I shows which counts the decoder uses when decoding the symbols from index 1 to index 16. The encoder must not only count the symbols in advance, but it also has to simulate the rescaling process. It cannot simply multiply the counts by 2 when a rescaling position is reached, because it is not known whether the new upscaled count has to be an odd or even number. This means that the encoder must not only store the final counts, but additionally the counts at each rescaling event.

The pseudo code of encoder and decoder is given in **Listing 2**. Compared to the procedure without rescaling, the encoder must maintain an array `countsR[][]` containing the symbol counts before a rescaling event takes place (at the decoder). Hence, the encoder simulates the decoder process in terms of counting the symbols and in terms of rescaling the counts (lines 21-30). Together with the symbol's counters, the positions of the rescaling events must also be stored (line 28), as this information is utilized during the actual encoding process. The number of rescaling events R should be set in advance to facilitate the dynamic allocation of array `countsR[][]`. Note that the rescaling interval must be smaller than the number N of symbols; otherwise, the index r is not set correctly in this pseudo code. The chosen integer division of the count values by two (bit-shift to the right) is only an example and other strategies could be used depending on the application.

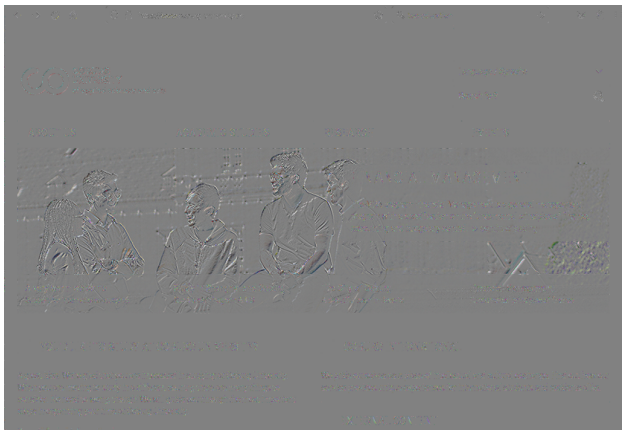
Listing 2. Pseudo code of adaptive LIFO coding with rescaling of symbols distribution. See text for details

```
N := number of symbols in sequence
K := number of different symbols in alphabet
R := number of rescaling events
sequ[] ... symbol sequence comprising N symbols
idxR[] ... symbol index at which rescaling takes place
count[] ... histogram of symbols, K elements
countR[][] ... counts before downscaling, RxK elements
rescaleInterval ... < N

for symbol = 0:K-1
{
  count[symbol] := 1; // initialize array
}

r := 0; // rescaling index
if encoder_flag
{
  for i = 0:N-1
  { // count symbols, simulate decoder process
    symbol := sequ[i];   // get current symbol
    if (i+1) % rescaleInterval == 0  // example event
    { // do the rescaling
      for symbol = 0:K-1
      {
        countR[r][symbol] := count[symbol]; // save count
        count[symbol] := count[symbol] >> 1; // downscale
      }
      idxR[r] := i; // save event location
      r := r + 1;
    }
    count[symbol] := count[symbol] + 1;
  }
  r := r - 1; // go back to last entry
  for i = N-1:-1:0
  { // encode symbols in reverse order
    symbol := sequ[i];   // get current symbol
    count[symbol] := count[symbol] - 1; // decrement
        corresponding count
    if i == idxR[r]
    { // position of rescaling event
      for symbol = 0:K-1
      {
        count[symbol] := countR[r][symbol]; // restore
      }
      if r > 0 { r := r - 1; }
    }
    encode( symbol, count); // encode symbol
  }
}
else // do the decoding
{
  for i = 0:N-1
  { // decode symbols in normal order
    symbol := decode( count); // decode symbol
    sequ[i] := symbol;   // put current symbol
    count[symbol] := count[symbol] + 1; // increment
        corresponding count
    if (i+1) % rescaleInterval == 0
    { // do the rescaling
      for symbol = 0:K-1
      {
        count[symbol] := count[symbol] >> 1; // downscale
      }
    }
  }
}
```



(a)

(b)



(c)

Fig. 1. Images of size 1785×1225 that have been used as input for coding tests: (a) the original image x, (b) prediction error image e_1 based on a simple prediction method, (c) prediction error image e_2 based on an advanced prediction method

IV. Investigations and Results

The described method of rescaling in combination with rANS coding has been extensively investigated. Here, the prosed scheme is exemplarily demonstrated using three image signals (**Figure 1**). The first is an original image (screenshot). The other two images are prediction-error signals obtained with two different prediction schemes. Each of the colour images has been converted into a byte stream so that all red intensities are sampled first, then all green intensities, and all blue signal values. This is also known as planar mode RRR..GGG..BBB. The rANS coding is additionally compared with the conventional arithmetic coding based on [19]. **Table II** contains the achieved compression ratios:

$$C_R = \frac{\text{number of bytes in raw format}}{\text{number of bytes in compressed file}}$$

Arithmetic coding (AC) and rANS coding have been tested in three different operating modes. The static mode determines the symbol distribution in advance and does not change it during the curse of coding. Since the decoder needs to know this distribution, it is transmitted as side information. The number of bits to be spent on transmission is adaptively chosen for each single count, resulting in about 600 additional

TABLE II
COMPRESSION RATIOS DEPENDING ON THE CHOSEN CODING METHOD AND THE MODE OF OPERATION.

	image x		image e_1		image e_2	
method	AC	rANS	AC	rANS	AC	rANS
static	1.614	1.618	3.472	3.494	4.843	4.878
adaptive	1.616	1.619	3.485	3.493	4.863	4.860
rescaling	2.527	2.522	4.389	4.400	6.085	6.097

bytes per image. The adaptive mode does not require this kind of transmission, because the coding process starts with counts equal to one (at the decoder). The third mode combines adaptive coding with periodic rescaling with a rescaling interval of 2^{12}. As can be seen, both, conventional arithmetic coding and rANS coding, achieve about the same compression performance and both significantly benefit from rescaling.

V. SUMMARY

The paper has explained and demonstrated a new method that enables adaptive processing in the context of rANS coding. Despite the diametric access of symbols (LIFO principle) rANS can not only be successfully combined with an adaptation of symbol counts, but a technique has also been developed that enables the rescaling of these counts. This opens up an area of applications for rANS coding in which fully-adaptive processing is a better option than static processing.

VI. ACKNOWLEDGEMENTS

This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 438221930.

REFERENCES

[1] D. Huffman, "A method for the construction of minimum redundancy codes," *Proc. of the IRE*, vol. 40, pp. 1098 – 1101, September 1952.
[2] J. Rissanen and G. Langdon, "Arithmetic coding," *IBM Journal of Research and Development*, vol. 23, pp. 149 – 162, March 1979.
[3] J. Duda, "Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding," January 2013. [Online]. Available: https://arxiv.org/abs/1311.2540
[4] J. Duda, K. Tahboub, N. J. Gadgil, and E. J. Delp, "The use of asymmetric numeral systems as an accurate replacement for huffman coding," in *2015 Picture Coding Symposium (PCS)*, 2015, pp. 65–69.
[5] Y. Collet, "Zstandard," accessed: January 12, 2023. [Online]. Available: https://facebook.github.io/zstd/
[6] "ISO/IEC 18181-1:2022: Information technology - JPEG XL image coding system - part 1: Core coding system." [Online]. Available: https://www.iso.org/standard/77977.html
[7] F. Giesen, "Interleaved entropy coders," 2014. [Online]. Available: https://arxiv.org/abs/1402.3392
[8] H. Yokoo and T. Shimizu, "Probability approximation in asymmetric numeral systems," in *2018 International Symposium on Information Theory and Its Applications (ISITA)*, October 2018, pp. 638 – 642.
[9] I. Blanes, M. Hernández-Cabronero, J. Serra-Sagristà, and M. W. Marcellin, "Redundancy and optimization of tANS entropy encoders," *IEEE Transactions on Multimedia*, vol. 23, pp. 4341–4350, 2021.
[10] D. Kosolobov, "The efficiency of the ans entropy encoding," 2022. [Online]. Available: https://arxiv.org/abs/2201.02514
[11] T. Alonso, G. Sutter, and J. E. L. De Vergara, "LOCO-ANS: An optimization of JPEG-LS using an efficient and low-complexity coder based on ans," *IEEE Access*, vol. 9, pp. 106 606–106 626, 2021.
[12] A. Moffat and M. Petri, "Ans-based index compression," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, November 2017, pp. 677 – 686.
[13] ——, "Index compression using byte-aligned ans coding and two- dimensional contexts," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, February 2018, pp. 405 – 413.
[14] M. Lettrich, "Fast and efficient entropy compression of alice data using ans coding," in *Proceedings of EPJ Web Conf*, vol. 245, November 2020.
[15] B. L. C. Barzen, F. Glazov, J. Geistert, and T. Sikora, "Accelerated deep lossless image coding with unified paralleleized GPU coding architecture," 2022. [Online]. Available: https://arxiv.org/abs/2207.05152
[16] J. Townsend, "A tutorial on the range variant of asymmetric numeral systems," 2020. [Online]. Available: https://arxiv.org/abs/2001.09186
[17] P. A. Hsieh and J.-L. Wu, "A review of the asymmetric numeral system and its applications to digital images," *Entropy*, vol. 24, no. 3, 2022. [Online]. Available: https://www.mdpi.com/1099-4300/24/3/375
[18] I. Witten, R. Neal, and J. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, pp. 520 – 540, Juni 1987.
[19] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited," in *Proceedings of the Data Compression Conference*, Snowbird, Utah, March 1995, pp. 202 – 211.