

Figure 10: The processing pipeline consists of a vanilla design with a DECIMAL XOR-based converter (reverter) and a specialized compressor (decompressor). An exception handler (gray section) is included to process failure cases.

**Algorithm 3** DeXOR\_Compressor ( $out, \delta$ , prefix  $\alpha_i$ , suffix  $\beta_i$ )

```

1:  $\bar{\ell}_i \leftarrow \lceil \delta \times \log_2(10) \rceil$  ▷ optimized binary suffix length
2: if  $\alpha_i = 0$  then ▷ using Lemma 3 (Sign Consistency)
3:    $out.writeBit(\beta_i < 0)$ 
4:  $out.write(abs(\beta_i), \bar{\ell}_i)$ 

```

**Algorithm 4** Decompressor (binary stream from storage stream  $in$ , values in global Buffer  $v'_{i-1}, \alpha_{i-1}, q_{i-1}, o_{i-1}$ )

```

1:  $code \leftarrow in.readInt(2)$ 
2: if  $code = 3$  then ▷ Exception Case [11]
3:   return Exception_Decompressor()
4: else
5:    $\alpha_i, q_i, o_i \leftarrow DeXOR\_Reverter(in)$ 
6:    $v'_i \leftarrow DeXOR\_Decompressor(in, \alpha_i, o_i, q_i)$ 
7:   Buffer  $\leftarrow v'_i, \alpha_i, q_i, o_i$ 
8:    $\omega_i \leftarrow decimal\_to\_binary(v'_i)$ 
9:   return  $\omega_i$ 
10: function DeXOR_Reverter( $in, code$ )
11:   if  $code = 0$  then
12:      $q_i \leftarrow in.readInt(5) - 20$ 
13:   else
14:      $q_i \leftarrow q_{i-1}$ 
15:   if  $code < 2$  then
16:      $o_i \leftarrow q_i + in.readInt(4)$ 
17:      $\alpha_i \leftarrow [v'_{i-1} \times 10^{-o_i}] \times 10^{o_i}$ 
18:   else
19:      $o_i \leftarrow o_{i-1}$ 
20:      $\alpha_i \leftarrow \alpha_{i-1}$  ▷ Acceleration of reconstruction
21:   return  $\alpha_i, q_i, o_i$ 
22: function DeXOR_Decompressor( $in, \alpha_i, o_i, q_i$ )
23:   if  $\alpha_i = 0$  then
24:      $sign \leftarrow in.readBoolean() ? -1 : 1$ 
25:   else
26:      $sign \leftarrow \alpha_i < 0 ? -1 : 1$ 
27:    $\bar{\ell}_i \leftarrow \lceil \delta \times \log_2(10) \rceil$  ▷  $\delta = k_i - q_i$ 
28:    $\beta_i \leftarrow sign \times in.readLong(\bar{\ell}_i)$ 
29:    $v'_i \leftarrow \alpha_i + \beta_i \times 10^{q_i}$ 
30:   return  $v'_i$ 

```

The efficient metadata storage we mentioned can not only save storage space during compression but can also be utilized to accelerate the computational process of decompression (Section 4.4, Algorithm 4). For example, the computation of reconstruction of the prefix  $\alpha_i$  (line 20 of Algorithm 4) can be omitted in the Case [10]. This is because when this case occurs ( $q_i = q_{i-1} \wedge o_i = o_{i-1}$ ), the following conclusion can be drawn:

$$\alpha_{i-1} = \alpha_i = [v_{i-1} \times 10^{-o_{i-1}}] \times 10^{o_{i-1}} = [v_i \times 10^{-o_i}] \times 10^{o_i}. \quad (8)$$

In the event of an exception, the storage of metadata (the tail coordinate  $q_i$  and the LCP coordinate  $o_i$ ) is skipped, and the process directly proceeds to the Exception Handler (see Section 5). The complete Exception handler module and its decompressor are presented in Algorithm 5 and Algorithm 6.

**Algorithm 5** Exception\_Handler ( $out, \omega_i, exp_{i-1}$  in buffer, EL and  $step$  are global variables)

```

1:  $exp_i \leftarrow (\omega_i >> 52) \& 0x7FFF$ 
2:  $ES_i \leftarrow exp_i - exp_{i-1}$  ▷ Exponential Subtraction
3:  $bias \leftarrow 2^{EL-1} - 1$ 
4: if  $ES_i \in [-bias, bias]$  then
5:    $out.write(ES_i + bias, EL)$ 
6:    $out.write(\omega_i < 0)$  ▷ 1-bit sign
7:    $out.write(\omega_i, 52)$  ▷ 52-bit fraction
8:   if  $ES_i \in [-\frac{bias-1}{2}, \frac{bias-1}{2}]$  then
9:      $step += 1$ 
10:  else
11:     $step \leftarrow 0$ 
12:  if  $EL > 1 \wedge step \geq \theta$  then ▷ Contraction
13:     $EL -= 1$ 
14:     $step \leftarrow 0$ 
15: else
16:    $out.write(bias, EL)$ 
17:    $out.write(\omega_i, 64)$ 
18:    $step \leftarrow 0$ 
19:   if  $EL < 10$  then ▷ Expansion
20:      $EL += 1$ 
21: return  $ES_i$ 

```

**Algorithm 6** Exception\_Decompressor (binary input stream  $in$  and the previous exceptional exponent  $exp_{i-1}$  in buffer, EL and  $step$  are global variables)

```

1:  $ES_i \leftarrow in.read(EL)$ 
2:  $bias \leftarrow 2^{EL-1} - 1$ 
3: if  $ES_i \in [-bias, bias]$  then
4:    $exp_i \leftarrow exp_{i-1} + ES_i$ 
5:    $sign \leftarrow in.readInt(1)$ 
6:    $\omega_i \leftarrow sign || exp_i || in.readLong(52)$ 
7:    $contract(ES_i, EL, step)$ 
8: else
9:    $\omega_i \leftarrow in.readDouble(64)$ 
10:   $exp_i \leftarrow get\_exp(\omega_i)$ 
11:   $step = 0$ 
12:   $expand(ES_i, EL)$ 
13: return  $\omega_i$ 

```

## A.4 Algorithm of Extension to Higher Buffer Size

Another attempt involves an extended buffer size to  $N$  strategy analogous to Chimp<sub>128</sub> but tailored specifically for DeXOR. What we need to do is to update the original results of LCP coordinates in Algorithm 1 and the reverter during decompression (Algorithm 4) with the method presented in Algorithm 7.

**Algorithm 7** Extension\_N (Buffer size  $N$ , Buffer:  $v'_{i-N}, \dots, v'_{i-1}$ , current value  $v'_i$ , tail coordinate  $q_i$ )

---

```

1: ...
2: LCP coordinate  $o_i \leftarrow \text{get\_LCP}(v'_i, q_i)$  ▷ line 3 of Algorithm 1
3:  $o_i \leftarrow \text{update\_LCP}(v'_i, o_i, N)$ 
4: ...
5:  $\text{out.write}(2 \text{ or } 1 \text{ or } 0, 2)$  ▷ line 6/8/11 of Algorithm 2
6:  $\text{out.write}(id_i - 1, \lceil \log_2 N \rceil)$ 
7: ...
8: function update_LCP( $v'_i, o_i, N$ ) ▷ Continue get_LCP
9:    $id_i \leftarrow 1$ 
10:  for  $j = 2$  to  $N$  do
11:    while  $\lfloor v'_i \times 10^{-(o_i-1)} \rfloor \times 10^{(o_i-1)} = \lfloor v'_{i-j} \times 10^{-(o_i-1)} \rfloor \times 10^{(o_i-1)}$  do
12:       $o_i \leftarrow o_i - 1$ 
13:       $id_i \leftarrow j$ 
14:  return  $o_i, id_i$ 
15: function DeXOR_Reverter_N( $in, q_i, code, N$ ) ▷ Replace DeXOR_Reverter
   (line 10 of Algorithm 4)
16:    $id_i \leftarrow \text{in.readInt}(\lceil \log_2 N \rceil) + 1$ 
17:   if  $code = 0$  then
18:      $q_i \leftarrow \text{in.readInt}(5) - 20$ 
19:   else
20:      $q_i \leftarrow q_{i-1}$ 
21:   if  $code < 2$  then
22:      $o_i \leftarrow q_i + \text{in.readInt}(4)$ 
23:   else
24:      $o_i \leftarrow o_{i-1}$ 
25:    $\alpha_i \leftarrow \lfloor v'_{i-id_i} \times 10^{-o_i} \rfloor \times 10^{o_i}$  ▷ Acceleration fails
26:  return  $\alpha_i, q_i, o_i$ 

```

---

The primary distinction from the original scheme lies in decompression. Under this extension, Equation 8 fails to hold, thereby precluding the acceleration of the reconstruction of the prefix  $\alpha_i$  (line 25 of Algorithm 7 and line 20 of Algorithm 4).

In fact, we have supplemented our work with experiments on  $N$  across different sizes and demonstrated that our algorithm achieves the most optimal compression when it is not extended ( $N = 1$ ). In the vast majority of cases, the extension does not yield better results than the original DeXOR due to the overhead associated with recording the index of the buffer.

## B SUPPORTING THEORY AND PROOFS

### B.1 Loss of Smoothness in Elf

We provide a formula to quantify the **smoothness loss** after conversion:

$$\hat{q} = \min(v_X.q, v_Y.q) \quad (9)$$

$$S(v_X, v_Y) = \text{CBL} \left( (v_X \times 10^{-\hat{q}}) \oplus (v_Y \times 10^{-\hat{q}}) \right), \quad (10)$$

The smoothness loss is then defined as:

$$\text{Loss} = S(\hat{v}_X, \hat{v}_Y) - S(v_X, v_Y). \quad (11)$$

Certain algorithms, such as Elf, introduce precision-related errors that disrupt smoothness. For Elf, the loss can be expressed as:

$$\text{Loss(Elf)} = S(v_X - \delta_{v_X}, v_Y - \delta_{v_Y}) - S(v_X, v_Y). \quad (12)$$

$\delta_{v_X}$  denotes the discrepancy between the value subsequent to erasure and the original value, which may be represented as:  $\delta_{v_X} = \text{sign} \times 2^{(\text{exp} - \text{bias})} \times \text{fraction}(\text{lower } g(v_X) \text{ bits})$ .

Lower  $g(v_X)$  bits are erased. A concept defined by the original author, Elf, but can be described in the language of this paper as:  $g(v_X) = 52 - (\lceil (-q) \log_2(10) \rceil + \text{exp} - \text{bias})$ .

When XOR operations are applied, precision mismatches occur. Specifically, when  $g(v_X) \neq g(v_Y)$ , it follows that  $\delta_{v_X} \neq \delta_{v_Y}$ . As a result, the Elf algorithm can disrupt up to:  $\text{abs}(g(v_X) - g(v_Y))$  bits of already-erased CBL.

Although Elf attempts to combine redundancy elimination and smoothness exploitation, examples illustrate that performing XOR after erasure can yield poor results. Reversing the order — erasure after XOR — is equally problematic. Moreover, precision becomes uncontrollable after XOR operations, making such optimizations challenging to implement effectively.

### B.2 Zero Loss of DECIMAL XOR Converter

We formally prove that the DECIMAL XOR converter incurs no smoothness loss.

**Lemma 5** (Zero Loss of DECIMAL XOR). *The smoothness loss of DECIMAL XOR satisfies:  $\forall v_X, v_Y \in \mathbb{R}, \text{Loss}(v_X \diamond v_Y) = 0$ .*

**PROOF.** According to the preconditions defined in Section 4.2.1, the DECIMAL XOR operation produces:

$$\hat{v}_X = v_X \diamond v_Y = v_X - \alpha, \quad \hat{v}_Y = v_Y - \alpha, \quad (13)$$

where  $\alpha$  denotes the shared prefix between  $v_X$  and  $v_Y$ .

Let  $\hat{q} = \min(v_X.q, v_Y.q)$ , we now compute the smoothness metric  $S(\hat{v}_X, \hat{v}_Y)$ :

$$\begin{aligned} S(\hat{v}_X, \hat{v}_Y) &= (\hat{v}_X \times 10^{-\hat{q}}) \oplus (\hat{v}_Y \times 10^{-\hat{q}}) \\ &= (v_X \times 10^{-\hat{q}} - \alpha) \oplus (v_Y \times 10^{-\hat{q}} - \alpha) \\ &= ((v_X \times 10^{-\hat{q}}) \oplus \gamma) \oplus ((v_Y \times 10^{-\hat{q}}) \oplus \gamma), \end{aligned}$$

where  $\gamma = -\alpha \times 10^{-\hat{q}}$ .

Since the  $\oplus$  operation is associative and  $\gamma$  cancels out due to shared prefixes, we have:

$$S(\hat{v}_X, \hat{v}_Y) = (v_X \times 10^{-\hat{q}}) \oplus (v_Y \times 10^{-\hat{q}}) = S(v_X, v_Y). \quad (14)$$

Thus, the smoothness loss is:

$$\text{Loss}(v_X \diamond v_Y) = S(\hat{v}_X, \hat{v}_Y) - S(v_X, v_Y) = 0. \quad (15)$$

This completes the proof.  $\square$

### B.3 Proof of Fixed Bit Allocation for Unsigned Binary Suffix

In Section 4.3, we introduce Lemma 4: For any  $\beta_i \in \mathbb{Z}$ , fixed allocation of  $\bar{\ell}_i$  bits achieves better compression than variable allocation of  $\ell_i$ , i.e.,  $\mathbb{E}[(4 + \bar{\ell}_i)] < \mathbb{E}[(6 + \ell_i)]$ . Its detailed proof is provided as follows:

**PROOF.** Given:  $\delta = o_i - q_i \in \mathbb{N}$ ,  $\text{abs}(\beta_i) \in [10^{\delta-1}, 10^\delta]$ .  $\ell_i = \lceil \log_2(\text{abs}(\beta_i) + 1) \rceil$ ,  $\bar{\ell}_i = \lceil \log_2(10^\delta) \rceil$ . We aim to prove:  $\mathbb{E}[(\bar{\ell}_i + 4)] < \mathbb{E}[(6 + \ell_i)] \iff \mathbb{E}[(\bar{\ell}_i - \ell_i - 2)] < 0$ .

We divide the range of  $\text{abs}(\beta_i)$  by powers of 2. Let  $j \in \mathbb{N}$  be the smallest integer such that  $2^j > 10^{\delta-1}$ , thus:  $2^{j-1} \leq 10^{\delta-1} < 2^j <$

$2^{j+1} < 2^{j+2} < 10^\delta < 2^{j+4}$ . The relationship between  $2^{j+3}$  and  $10^\delta$  gives two complementary cases:

**Case (1)**  $2^{j+3} > 10^\delta$  with possibility  $\mathbb{P}_1$ ; fixed allocation  $\bar{\ell}_i = \lceil \log_2(10^\delta) \rceil = j + 3$  whereas

$$\ell_i = \begin{cases} j & \text{if } \text{abs}(\beta_i) \in [10^{\delta-1}, 2^j), \\ j+1 & \text{if } \text{abs}(\beta_i) \in [2^j, 2^{j+1}), \\ j+2 & \text{if } \text{abs}(\beta_i) \in [2^{j+1}, 2^{j+2}), \\ j+3 & \text{if } \text{abs}(\beta_i) \in [2^{j+2}, 10^\delta). \end{cases}$$

Calculating expectation of case (1):  $\mathbb{E}_1 = \mathbb{E}(\bar{\ell}_i + 4 \mid 2^{j+3} > 10^\delta) - \mathbb{E}(\ell_i + 6 \mid 2^{j+3} > 10^\delta) = j + 1 - \mathbb{E}(\ell_i \mid 2^{j+3} > 10^\delta)$ .

$$\begin{aligned} \mathbb{E}(\ell_i \mid 2^{j+3} > 10^\delta) &= \frac{(j+3)(10^\delta - 2^{j+2})}{10^\delta - 10^{\delta-1}} + \frac{(j+2)(2^{j+2} - 2^{j+1})}{10^\delta - 10^{\delta-1}} \\ &\quad + \frac{(j+1)(2^{j+1} - 2^j)}{10^\delta - 10^{\delta-1}} + \frac{j(2^j - 10^{\delta-1})}{10^\delta - 10^{\delta-1}} \\ &= j + \frac{3(10^\delta - 2^{j+2}) + 2(2^{j+2} - 2^{j+1}) + (2^{j+1} - 2^j)}{10^\delta - 10^{\delta-1}} \\ &= j + \frac{3 \times 10^\delta - 2^{j+2} - 2^{j+1} - 2^j}{10^\delta - 10^{\delta-1}} \\ &= j + 3 - \frac{2^{j+2} + 2^{j+1} + 2^j - 3 \times 10^{\delta-1}}{10^\delta - 10^{\delta-1}} \\ &> j + 3 - \frac{2^{j+2} + 2^{j+1} + 2^j - 3 \times 2^{j-1}}{2^{j+2} - 2^j} \\ &= j + 3 - \frac{(8 + 4 + 2 - 3) \times 2^{j-1}}{(8 - 2) \times 2^{j-1}} \\ &= j + 3 - \frac{11}{6} = j + \frac{7}{6} \end{aligned}$$

$$\Rightarrow \mathbb{E}_1 = j + 1 - \mathbb{E}(\ell_i \mid 2^{j+3} > 10^\delta) < -\frac{1}{6}.$$

Calculating possibility  $\mathbb{P}_1$  of case (1):  $\mathbb{P}_1\{2^{j+3} > 10^\delta\} = \mathbb{P}\{(j+3) \log_{10} 2 > \delta\}$ ,

we know the condition  $2^j \geq 10^{\delta-1} \Rightarrow j \log_{10} 2 + 1 \geq \delta$ ,

and we know  $2^{j-1} \leq 10^{\delta-1} \Rightarrow (j-1) \log_{10} 2 + 1 \leq \delta$ ,

so  $\delta \in [(j-1) \log_{10} 2 + 1, j \log_{10} 2 + 1]$

$$\begin{aligned} \mathbb{P}\{(j+3) \log_{10} 2 > \delta\} &= \frac{(j+3) \log_{10} 2 - (j-1) \log_{10} 2 - 1}{\log_{10} 2} \\ &= \frac{4 \log_{10} 2 - 1}{\log_{10} 2} \Rightarrow \mathbb{P}_1\{2^{j+3} > 10^\delta\} = 4 - \log_2 10 \approx 0.6781. \end{aligned}$$

**Case (2)**  $2^{j+3} \leq 10^\delta$  with possibility  $\mathbb{P}_2\{2^{j+3} \leq 10^\delta\} = 1 - \mathbb{P}_1 \approx 0.3219$ ; fixed allocation  $\bar{\ell}_i = \lceil \log_2 10 \rceil = \lceil \log_2(2^{j+4}) \rceil = j + 4$  whereas

$$\ell_i = \begin{cases} j & \text{if } \text{abs}(\beta_i) \in [10^{\delta-1}, 2^j), \\ j+1 & \text{if } \text{abs}(\beta_i) \in [2^j, 2^{j+1}), \\ j+2 & \text{if } \text{abs}(\beta_i) \in [2^{j+1}, 2^{j+2}), \\ j+3 & \text{if } \text{abs}(\beta_i) \in [2^{j+2}, 2^{j+3}), \\ j+4 & \text{if } \text{abs}(\beta_i) \in [2^{j+3}, 10^\delta). \end{cases}$$

The expectation of this case:  $\mathbb{E}_2 = \mathbb{E}(\bar{\ell}_i + 4 \mid 2^{j+3} < 10^\delta) - \mathbb{E}(\ell_i + 6 \mid 2^{j+3} < 10^\delta) = j + 2 - \mathbb{E}(\ell_i \mid 2^{j+3} < 10^\delta)$ .

$$\begin{aligned} \mathbb{E}(\ell_i \mid 2^{j+3} < 10^\delta) &= j + \frac{4 \times 10^\delta - 2^{j+3} - 2^{j+2} - 2^{j+1} - 2^j}{10^\delta - 10^{\delta-1}} \\ &= j + 4 - \frac{2^{j+3} + 2^{j+2} + 2^{j+1} + 2^j - 4 \times 10^{\delta-1}}{10^\delta - 10^{\delta-1}} \\ &> j + 4 - \frac{2^{j+3} + 2^{j+2} + 2^{j+1} + 2^j - 4 \times 2^{j-1}}{2^{j+3} - 2^j} \\ &= j + 4 - \frac{(16 + 8 + 4 + 2 - 4) \times 2^{j-1}}{(16 - 2) \times 2^{j-1}} \\ &= j + \frac{13}{7} \end{aligned}$$

$$\Rightarrow \mathbb{E}_2 = j + 2 - \mathbb{E}(\ell_i \mid 2^{j+3} < 10^\delta) < \frac{1}{7}.$$

Finally, we have the overall expectation  $\mathbb{E}[(\bar{\ell}_i - \ell_i - 2)] = \mathbb{E}_1 \times \mathbb{P}_1 + \mathbb{E}_2 \times \mathbb{P}_2 < -\frac{1}{6} \times \mathbb{P}_1 + \frac{1}{7} \times \mathbb{P}_2 \approx -0.067 < 0$ .  $\square$

## C MORE EXPERIMENTAL DETAILS

### C.1 Datasets Description

**Time-Series Datasets:**

- City-temp (CT)<sup>7</sup>: Temperature records from major cities worldwide, collected by the University of Dayton.
- NEXO Datasets: A collection of five datasets from various sensors, published by the National Ecological Observatory Network (NEON):
  - Air-pressure (AP)<sup>8</sup>,
  - Dewpoint-temperature (DPT)<sup>9</sup>,
  - IR-bio-temperature (IR)<sup>10</sup>,
  - PM10-dust (PM)<sup>11</sup>,
  - Wind-speed (WS)<sup>12</sup>.
- Stock Exchange Datasets<sup>13</sup>: Exchange price data from three countries:
  - UK (Stocks-UK, SUK),
  - USA (Stocks-USA, SUSA),
  - Germany (Stocks-DE, SDE).
- Meteoblue Datasets<sup>14</sup>: Historical weather data for Basel, Switzerland, including:
  - Wind speed (Basel-wind, BW),
  - Temperature (Basel-temp, BT).
- InfluxDB Datasets<sup>15</sup>: A set of datasets from various domains, including:
  - Air-sensor (AS),
  - Bird-migration tracking (BM),
  - Bitcoin-price (BP).

<sup>7</sup>2023. Daily Temperature of Major Cities. <https://www.kaggle.com/sudalairajkumar/daily-temperature-of-major-cities>

<sup>8</sup>2022. Barometric pressure. <https://data.neonscience.org/data-products/DP1.00004.001/RELEASE-2022>

<sup>9</sup>2022. Relative humidity above water on-buoy. <https://data.neonscience.org/data-products/DP1.20271.001/RELEASE-2022>

<sup>10</sup>2022. IR biological temperature. <https://data.neonscience.org/data-products/DP1.00005.001/RELEASE-2022>

<sup>11</sup>2022. Dust and particulate size distribution. <https://data.neonscience.org/data-products/DP1.00017.001/RELEASE-2022>

<sup>12</sup>2022. 2D wind speed and direction. <https://data.neonscience.org/data-products/DP1.00001.001/RELEASE-2022>

<sup>13</sup>2020. Financial data set used in INFORE project. [https://zenodo.org/records/3886895#Y4DdzHZByM\\_](https://zenodo.org/records/3886895#Y4DdzHZByM_)

<sup>14</sup>2023. Basel Historical Weather Data. [https://www.meteoblue.com/en/weather/archive/export/basel\\_switzerland](https://www.meteoblue.com/en/weather/archive/export/basel_switzerland)

<sup>15</sup>2013. Scalable datastore for metrics, events, and real-time analytics. <https://github.com/influxdata/influxdb>

### Non-Time-Series Datasets:

- Electric vehicle charging sessions (EVC)<sup>16</sup>.
- Global food prices (FP) for December 2020<sup>17</sup>.
- Bitcoin transaction values (Blockchain transactions, BL)<sup>18</sup> for a specific day.
- Storage disk benchmarking scores (SSD)<sup>19</sup>.
- Geographic city coordinates<sup>20</sup>, which include:
  - Cartesian coordinates (CA),
  - Longitude and latitude coordinates (CO).
- Position of Interest (POI)<sup>21</sup> radian coordinates extracted from Wikipedia parsing, including:
  - Angular coordinates (PA),
  - Polar coordinates (PO).

## C.2 Regularity Analysis: Case Proportions and Efficiency

Based on the reuse relationships between the tail coordinates  $q_i$ , the LCP coordinates  $k_i$ , and their historical values  $q_{i-1}$  and  $k_{i-1}$ , we categorize the scenarios into four distinct cases. We conduct experiments across all existing datasets and calculate the average proportion of each distinct case. The results are illustrated in Figure 11, where we compute the average proportions of these cases across all datasets.

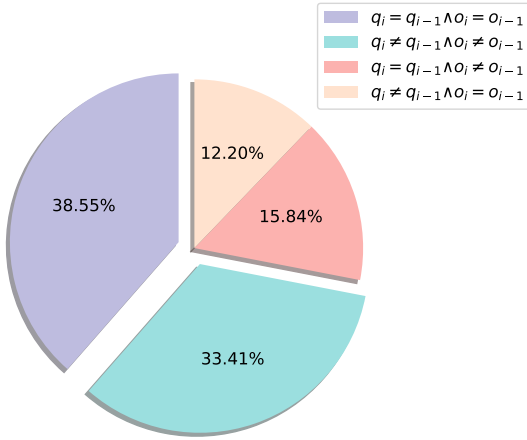


Figure 11: Proportion of reuse cases described in Section 4.2.2.

Referring to Figure 11, it is evident that Case 110 (see Section 4.2.2), which achieves the maximum efficiency by occupying only 2 bits, accounts for the largest proportion, at 38.55%. The second most significant case, Case 011 (occupying 6 bits), constitutes 33.41%. Together, these two dominant cases cover the majority of the scenarios.

<sup>16</sup>2023. Electric Vehicle Charging Dataset. <https://www.kaggle.com/datasets/michaelbryantds/electric-vehicle-charging-dataset>

<sup>17</sup>2021. Global Food Prices Database (WFP). <https://data.humdata.org/dataset/wfp-food-prices>

<sup>18</sup>2023. Bitcoin Transactions. <https://gz.blockchair.com/bitcoin/transactions/>

<sup>19</sup>2022. SSD and HDD Benchmarks. <https://www.kaggle.com/datasets/alanjo/ssd-and-hdd-benchmarks>

<sup>20</sup>2023. World City. <https://www.kaggle.com/datasets/kuntalmaity/world-city>

<sup>21</sup>2023. Points of Interest POI Database. <https://www.kaggle.com/datasets/ehallmar/points-of-interest-poi-database>

The remaining two cases contribute a combined proportion of only 28.04%. Notably, the scenario where  $q_i \neq q_{i-1} \wedge k_i = k_{i-1}$ , despite its multiplexing effect (occupying 7 bits), represents the smallest proportion, at merely 12.20%. To optimize efficiency, we merge these cases into Case 010, where  $q_i \neq q_{i-1}$ . This merging allows us to implement an optimal allocation scheme while reserving Case 111 for the Exception Handler module (see Section 5).

## C.3 Comparisons of Exponential Subtraction and Exponential xor

To validate the effectiveness of xor and subtraction operations in managing high-precision datasets, we conducted experiments on three datasets: AS, PA, and PO. The results of these experiments are presented in Table 7.

As shown in Table 7, the overall efficacy of the two approaches is comparable. However, the subtraction method tends to produce a lower number of CBL in certain cases, particularly for the PO dataset. Specifically, while xor demonstrates better performance for PA, subtraction outperforms in PO, with both methods performing equally well on AS.

Table 7: CBL generated by different operations.

Operation	AS	PA	PO
$exp_i \oplus exp_{i-1}$	0.02	0.51	2.23
$exp_i - exp_{i-1}$	0.02	0.65	1.14

## C.4 Compression with Prior Precision Judgment

As discussed in Section 5.3, we explore additional applications of the Exception Handler module. Specifically, we investigate its capability to independently compress data without involving the main process of the DeXOR, provided that prior judgments on data precision are available. Such scenarios are common in many practical applications.

To validate the effectiveness of the Exception Handler in this context, we conduct supplementary experiments on three high-precision datasets: AS, PA, and PO. The results, presented in Table 8, compare the performance of DeXOR with the Exception Handler used independently.

Table 8: Comparison with schemes with prior precision judgment (Exception Handler only).

Dataset	AS		PA		PO	
	DeXOR	Exception	DeXOR	Exception	DeXOR	Exception
ACB	52.28	54.12	57.86	56.10	58.70	56.86
Comp. Speed	7.45	4.83	2.14	8.16	1.86	10.20
Decomp. Speed	62.16	22.63	59.51	48.61	24.59	53.81

The results reveal several important insights:

- **Compression Ratio:** The Exception Handler alone achieves marginal gains in compression ratio for certain datasets (e.g., PA and PO), primarily due to the omission of the 2-bit case code. However, when excluding this gain, the compression ratio of the Exception Handler is consistently inferior to that of the complete DeXOR. This gap is particularly pronounced in the time-series dataset AS, where the main process of DeXOR demonstrates significantly better performance by effectively reducing precision.



Table 9: A detailed comparison of the DeXOR with other algorithms that have higher buffer sizes.

Datasets		Time-Series Datasets with Ascending $dp$														Non-Time-Series Datasets with Ascending $dp$										GEOMEAN	
		WS	PM	CT	IR	DPT	SUSA	SUK	SDE	AP	BM	BW	BT	BP	AS	FP	EVC	SSD	BL	CA	CO	PA	PO	FULL	low $dp$		
ACB	DeXOR	10.35	7.12	11.33	8.01	13.22	9.71	11.59	12.22	14.87	19.47	30.67	29.25	25.89	52.28	12.38	14.13	13.27	15.00	24.76	26.53	57.86	58.70	17.82	12.70		
	Chimp <sub>128</sub>	13.96	13.21	25.28	16.39	27.70	17.56	26.97	21.37	47.59	37.27	48.07	33.83	55.61	58.35	24.41	24.86	17.13	36.31	51.77	55.02	60.53	64.42	31.37	24.14		
	SELF*	11.61	8.40	12.12	8.82	15.20	10.51	11.08	11.94	16.30	21.80	32.90	30.82	29.56	50.57	13.25	17.13	13.56	17.77	29.51	34.40	59.36	65.23	19.55	14.16		
	ALP	25.87	62.06	50.48	42.95	51.50	41.47	41.97	48.21	48.30	53.01	59.54	61.97	58.71	71.32	49.77	61.43	47.79	49.73	56.04	53.16	80.57	81.72	52.97	47.66		
	Elf*	8.80	6.35	10.89	7.51	12.83	8.71	9.21	10.97	14.45	19.88	30.55	28.88	27.07	49.22	12.41	15.09	13.32	16.45	27.57	32.86	57.62	63.15	17.55	12.34		
Comp. Speed	DeXOR	23.86	37.08	32.88	51.57	16.74	49.63	37.71	19.60	33.22	3.61	4.65	21.44	13.49	7.45	15.89	6.77	19.00	21.82	19.93	17.49	2.14	1.86	14.94	24.05		
	Chimp <sub>128</sub>	18.86	26.44	9.57	23.60	4.68	42.96	32.22	30.16	31.88	3.58	14.26	24.91	13.77	1.73	5.42	10.36	1.05	34.46	30.41	29.54	9.41	4.85	12.65	16.62		
	SELF*	15.81	28.63	26.85	29.26	16.48	60.10	39.06	17.54	30.53	3.78	3.77	20.33	47.89	3.79	20.13	7.26	24.73	20.33	22.89	19.12	1.57	1.94	14.54	22.53		
	ALP	0.82	0.90	2.48	1.96	2.11	2.97	5.13	1.53	3.37	0.67	1.47	5.24	7.82	4.37	1.60	0.17	2.44	3.34	3.68	3.77	1.62	1.76	2.07	1.87		
	Elf*	19.04	30.42	34.17	51.20	20.08	53.67	36.06	16.74	32.74	3.51	4.88	17.97	40.40	8.56	16.42	10.39	17.82	23.41	18.75	19.67	2.18	2.54	16.06	24.34		
Dec. Speed	DeXOR	48.28	47.15	63.86	105.21	56.43	84.58	87.13	91.15	92.65	8.89	40.85	68.78	26.22	62.16	62.48	27.25	82.46	42.06	73.83	64.61	59.51	24.59	53.12	63.29		
	Chimp <sub>128</sub>	75.87	79.22	14.58	70.45	11.26	60.12	55.77	52.42	56.42	6.94	34.28	51.93	52.02	25.02	67.52	43.03	20.40	64.15	44.63	55.45	50.31	38.86	40.27	42.97		
	SELF*	32.56	33.19	55.52	81.44	53.35	70.14	69.83	84.81	106.25	8.09	41.69	87.06	33.93	61.26	75.22	29.35	62.06	54.13	77.76	42.97	56.28	21.26	49.69	54.30		
	ALP	39.80	22.74	32.64	64.32	26.71	43.11	43.02	53.06	152.04	6.69	32.31	58.89	18.49	67.95	38.59	20.73	71.81	35.97	61.13	57.12	48.67	20.31	38.73	41.03		
	Elf*	40.67	31.17	64.35	5.24	57.07	76.88	65.65	60.82	130.65	6.60	33.19	74.21	2.93	5.31	73.74	1.23	6.15	46.54	78.70	2.62	51.41	19.63	23.41	23.14		

- **Compression and Decompression Speed:** The Exception Handler exhibits considerable improvements in compression speed and modest gains in decompression speed. For non-time-series datasets, such as PA and PO, the Exception module is notably faster. However, it struggles with time-series data, such as AS, where the main process retains its advantage.

These findings suggest that the independent use of the Exception Handler is more suitable for high-precision, non-time-series datasets that rarely require the precision reduction capabilities of the main process. While the Exception Handler provides speed advantages, the main process of DeXOR continues to outperform it in terms of compression effectiveness for time-series data.

### C.5 Handling Extreme Cases with the Exception Handler

This section further validates the effectiveness of the Exception module on high-precision datasets: AS, PA, and PO. As illustrated in Figure 12, we compare DeXOR against an ablated version that excludes the Exception Handler. These datasets are characterized by virtually no representational redundancy, making them challenging for compression algorithms.

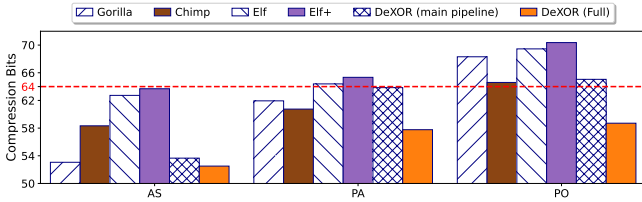


Figure 12: Comparison of DeXOR with and without the Exception Handler module. The red line ( $y = 64$ ) indicates the average bit occupancy of double precision data without compression.

For such datasets, only algorithms that fully exploit temporal smoothness, such as XOR-based compressors like Gorilla and Chimp/Chimp<sub>128</sub>, achieve any level of compression. In contrast, algorithms like Camel, Elf/Elf+, and the ablated DeXOR incur overheads that outweigh their compression gains.

The extreme PO dataset, which lacks both precision redundancy and temporal smoothness, poses a significant challenge. In this scenario, all algorithms fail except DeXOR with its Exception Handler. The Exception Handler adapts by capturing the largest residual similarities at minimal cost. Even in the theoretical worst-case scenario, where consecutive values differ by more than  $2^{1023}$ , the Exception Handler incurs at most a one-bit penalty.

Under realistic conditions, the Exception Handler consistently secures the best compression ratios for high-precision datasets. Its ability to adaptively handle extreme cases ensures that DeXOR remains effective, even on datasets with minimal redundancy. These results highlight the critical role of the Exception Handler in achieving superior performance on challenging high-precision data.

### C.6 More Details of Higher Buffer Size Schemes

In Table 9, we present a more detailed comparison of the results. In terms of the geometric mean, DeXOR demonstrates the fastest decompression speed and the second-highest compression ratio and speed, trailing only Elf\*. When compared with the streaming version of Elf\*, namely SELF\*, DeXOR outperforms it across all metrics. An intriguing observation is that while Elf\* exhibits an advantage in the compression ratio for time-series data, DeXOR generally achieves a superior compression ratio for non-time-series data.