改动一位，节省字节：浮点时间序列数据的压缩

*

Francesco Taurone、Daniel E. Lucani、Marcell Feh´er 和 Qi Zhang DIGIT，电气与计算机工程系 奥胡斯大学{francesco.taurone、daniel.lucani、sw0rdf1sh、qz}@ece.au.dk

2023年3月9日

arXiv:2303.04478v1 [cs.DS] 8 Mar 2023

摘要

预计物联网设备的数量在未来几年将继续大幅增长，与此同时，需要传输、处理和存储的数据量也将不断增加。支持直接对压缩数据进行分析的压缩技术，有望为系统高效扩展以应对这些日益增长的需求铺平道路。本文提出了两种新颖的方法，用于对浮点数据流进行预处理，以提升各种物联网数据压缩器的压缩能力。特别是，研究表明，这些技术对于近期支持随机访问和分析、同时保持良好压缩效果的压缩器十分有益。在允许最多1%的恢复误差的情况下，我们的技术可将压缩后的数据量最多减少80%。

1 引言

物联网设备会生成大量需要传输、存储和分析的数据。压缩数据块的优势包括降低数据传输成本、更高效地利用带宽，以及减少数据存储需求。我们重点关注时间序列数据压缩算法 [1], ，其最常见的目标是以较少的内存和计算能力，生成、分包、压缩并存储连续数据流，这些数据流可以来自物联网、金融领域或其他应用场景。

本文聚焦于使用压缩算法之前的数据处理。我们的目标是修改各个样本，使其比原始数据流更适合压缩。具体而言，我们提出了两种新颖的变换：加法和乘法变换。我们在图1中对其进行了总结：假设输入为一组一维样本 $[x_1, \ldots, x_n]$，应用这些变换后，我们得到一个数据集 $[y_1, \ldots, y_n]$。我们逐个处理数值，利用浮点数表示的特性（其基本结构如图2所示），以适用于随机访问压缩器，例如广义去重 [2]。我们选择每种变换的参数，使所有 y_i 在相同位置都具有一些相同的位（例如，表1最后一行中，最后21个尾数位均为零）。在数据恢复过程中，应用逆变换后得到$[\tilde{x_1}, \ldots, \tilde{x}_n]$，即重建数据集。尽管我们的方法是有损的，但用户可以为数据流中的每个样本指定最大恢复误差（即失真）的上限。允许的最大误差越大，压缩的潜力就越大。

1.1 动机示例

假设有两个数 x_1 和 x_2，如表 1 所示。尽管它们的十进制形式有许多相同的数字，但它们的尾数只有 2 位相同。首先应用加法

*本研究得到 IoTalentum 项目的支持。该项目属于玛丽·斯克沃多夫斯卡-居里行动下的创新培训网络（ITN）—欧洲培训网络（ETN），由欧盟“地平线 2020”研究与创新计划资助，项目编号为 953442。-

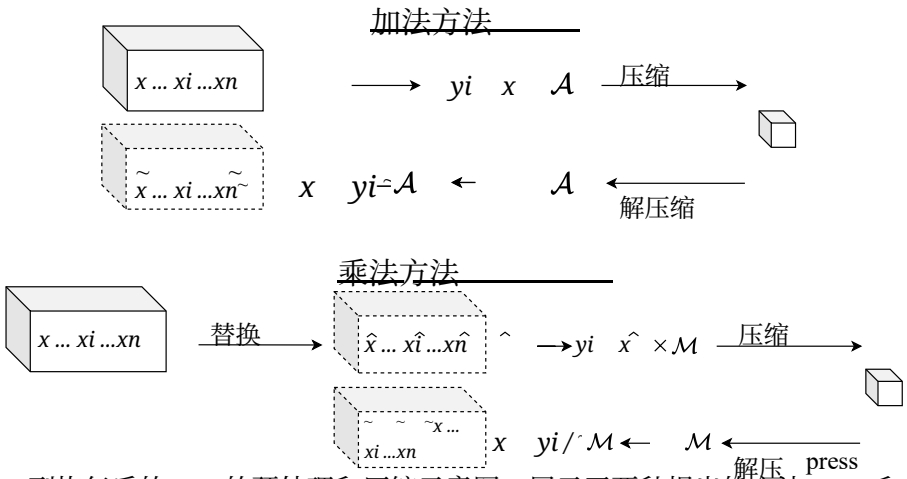

图1：从原始 x_i 到恢复后的 $\tilde{x}_i$ 的预处理和压缩示意图，展示了两种提出的方法。 x_i 和 $\tilde{x}_i$ 可能彼此不同。

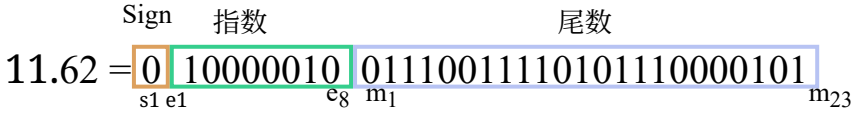

图2：32 位 IEEE 754 浮点数的编码。

变换时，我们选择一个参数 $\mathcal{A}$，将其加到数据流中的所有数据上；在本例中，这些数据分别表示为 x_1 和 x_2。这里我们使用 $\mathcal{A} = 1738.0$，得到 $y_1 = 1791.333$ 和 $y_2 = 2047.333$。虽然这种变换看似简单，却使 y_1 和 y_2中有多位数值相同，因此压缩效果可以比原始数据更好。此外，这两个数的指数现在也相同。

对于乘法变换，我们首先谨慎地修改比特流，在尾数中创建模式；当这些模式乘以特定的 $\mathcal{M}$ 值时，会在结果尾数中生成一串零。第一次比特模式变换从 x_1 输出 $\hat{x}_1$，并从 x_2 输出 $\hat{x}_2$。然后，我们乘以 $\mathcal{M} = 3.0$ 以生成 y_1 和 y_2。我们注意到，尾数中的全部 23 位都是共有的，因此可以实现更有效的压缩。

1.2 相关工作

通过利用数据结构及其上下文含义来降低压缩后的数据量，已有大量相关研究，例如针对图像、音频和时间序列数据的研究 [1]。压缩的关键步骤之一是对数据集进行初始处理，目的是清理数据，并筛除可能妨碍压缩的噪声、异常值或其他不需要的成分。对于来自物联网设备的数据，这一过程尤为重要，因为传感器可能噪声较大，且会随着时间推移而性能退化 [3]。通常需要依靠用户的知识和经验来为压缩准备数据。另一种互补方法通常是压缩器流水线的一部分，即使用采样系统的预测器或模型来表示相同的信息 [4]。或者，我们可以改变信号域，以编码其他特征，例如通过离散余弦变换和离散小波变换转换到频域 [5]。其中一些技术作为专用压缩器流水线的一部分，应用于特定类型的数据集时效果良好。我们提出了两种用途更广泛的预处理方法，在将浮点时间序列流作为输入传递给多种可能的压缩器之前对其进行变换。它们通过增加数据集中共享的比特值数量来降低熵，从而提高压缩器的效率。Kl¨ower 等人提出的预处理方法也旨在实现这一目标，该方法发表于*Nature Computational Science* [6], ，我们将其用作评估所提技术的基准。

表 1：预处理前后

原始	浮点数表示（符号位.指数.尾数）
$x_1 = 53.333$	$1{\cdot}2^{5}{\cdot}[1.10101010101010101010100]_{\text{base2}}$
$x_2 = 309.333$	$1{\cdot}2^{8}{\cdot}[1.00110101010101010101011]_{\text{base2}}$
公共位：	xx
加法变换	**，$\boldsymbol{A = 1738.0}$**
$x_1 + \mathcal{A} = 1791.333$	$1{\cdot}2^{10}{\cdot}[1.10111111110101010101000]_{\text{base2}}$
$x_2 + \mathcal{A} = 2047.333$	$1{\cdot}2^{10}{\cdot}[1.11111111110101010101000]_{\text{base2}}$
公共位：	x x xxxxxxxxxxxxxxxxxxxx
乘法变换，$\boldsymbol{\mathcal{M} = 3.0}$	
$\hat{x}_1 = 53.33333206176758, \hat{x}_2 = 309.3333435058594$	
$\hat{x}_1 \cdot \mathcal{M} = 160.0$	$1{\cdot}2^{7}{\cdot}[1.01000000000000000000000]_{\text{base2}}$
$\hat{x}_2 \cdot \mathcal{M} = 928.0$	$1{\cdot}2^{9}{\cdot}[1.01000000000000000000000]_{\text{base2}}$
公共位：	xxxxxxxxxxxxxxxxxxxxxxx

2 背景

2.1 性能指标

我们的最终目标是证明，这些技术能够减小输出文件的大小，从而提升压缩性能。我们使用压缩比（CR）作为衡量指标，其定义如下：

$$\text{CR} = \frac{\text{size of compressed data}}{\text{size of uncompressed data}}. \tag{1}$$

我们还需要一个名为最大恢复误差的参数，以便用户能够设定恢复误差上限，而该上限最终取决于应用需求。我们通过同时以绝对误差的形式定义该参数来实现这一点

$$E^{\Delta} = \max_i \Delta_i \quad \text{with } \Delta_i = |\tilde{x}_i - x_i|, \tag{2}$$

以及相对于原始值的相对误差

$$E^{\delta} = \max_i \delta_i \quad \text{with } \delta_i = (|\frac{\tilde{x}_i - x_i}{x_i}|). \tag{3}$$

2.2 浮点数（FP）

尽管表示实数和有理数 [7], 的方法多种多样，我们聚焦于标准 IEEE-754 [8]。虽然我们的描述和实验采用 32 位版本，但所提出的变换很容易适配该标准中的较小或扩展精度格式，例如 16、64 和 128 位。给定一个 FP 数 n，其 32 位结构如图 2 所示，分为三个部分：

- 符号位 *'S'*： $\{s_1\}$ 位。'0' 表示 $n \geq 0$，'1' 表示 $n < 0$。
- 指数 *'E'*： $\{e_1 \dots e_8\}$ 位。它采用偏置表示，这意味着从中减去偏置 $b = 2^{k-1} - 1 = 127$ 后，将其解释为无符号整数，其中 k 是指数位的数量。
- 尾数 *'M'*： $\{m_1 \dots m_i \dots m_{23}\}$ 位。每个 m_i 表示 2^{E-b-i}。

为了将位转换为数字，我们使用以下方程

$$\begin{aligned} n &= (-1)^S \cdot 2^{E-b} \cdot (1 + M \cdot 2^{-23}) \\ &= (-1)^S \cdot [2^{E_U} + m_1 \cdot 2^{E_U - 1} + \dots + m_{23} \cdot 2^{E_U - 23}] \end{aligned} \tag{4}$$

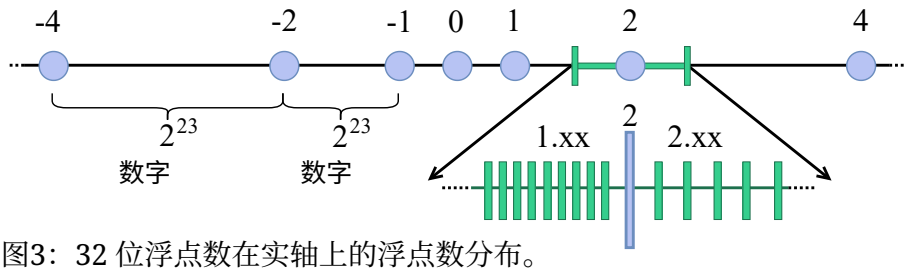

图3：32 位浮点数在实轴上的浮点数分布。

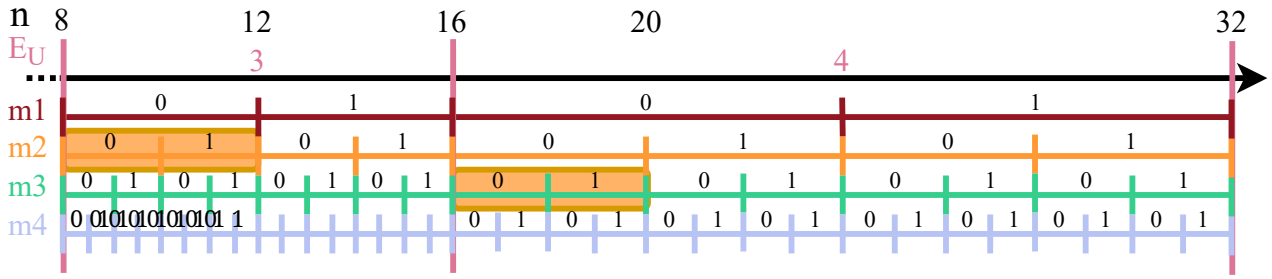

图4：具有 $x_i \in [8, 12]$ 的数据集（范围为橙色）。通过应用 $y_i = x_i + 8$，我们将数据集移至区域 $[16, 20]$, ，此处的数字具有更大的指数（$E_U = 4$，而原始值为 $E_U = 3$）。因此，移位后的数据集中的每个数字都将具有 $m_2 = 0$。

其中 $E_U = E - b \neq 0$。对于每个 n，其表示中可使用的最小量是 $2^{E_U - 23}$，这取决于该数字的指数。我们可以将此精度定义为

$$P(E_U) = 2^{E_U - 23}. \tag{5}$$

因此，在精度为 $P(E_U)$ 的区域内，两个相邻的可由浮点数表示的实数之差为 $P(E_U)$，更正式地说

$$|n_1 - n_2| \geq P(E_U), \forall n_1 \neq n_2 \in [2^{E_U}; 2^{E_U+1}]. \tag{6}$$

由于两个相邻的可由浮点数表示的数之间的距离取决于其指数，即 E_U，因此离零越远，实轴上的浮点数就越稀疏，如图 3 所示。当 $n \in [0, 2^{-b}[$, 且 $b = 127$ （对于 32位浮点数）时， $E = 0$ ，该数称为次正规数。标准为表示和处理这些极小且罕见的值规定了特定规则，以下内容中我们不会讨论这些值。唯一的例外是 $n = 0.0$，拟议的变换会单独处理它。

3 预处理方法

3.1 加法变换

此方法可概括为以下操作：

$$y_i = x_i + \mathcal{A}, \quad \forall x_i \in \mathscr{D} \tag{7}$$

其中， $\mathcal{A} > 0$ 称为加法参数， $\mathscr{D}$ 是数据集。其思路是将数据集中的所有样本平移到实轴上的某个合适区域。如图 4 所示，通过策略性地选择 $\mathcal{A}$ ，我们可以确保加法变换后的数据集中的所有数都具有相同的指数和若干公共尾数位。这样做的代价是，样本 x_i 从其原始区域的精度变为新区域 y_i的精度时，可能会产生恢复误差。为了恢复原始数据，我们应用逆变换 $\tilde{x}_i = y_i - \mathcal{A}$，因此需要将 $\mathcal{A}$ 作为元数据存储。

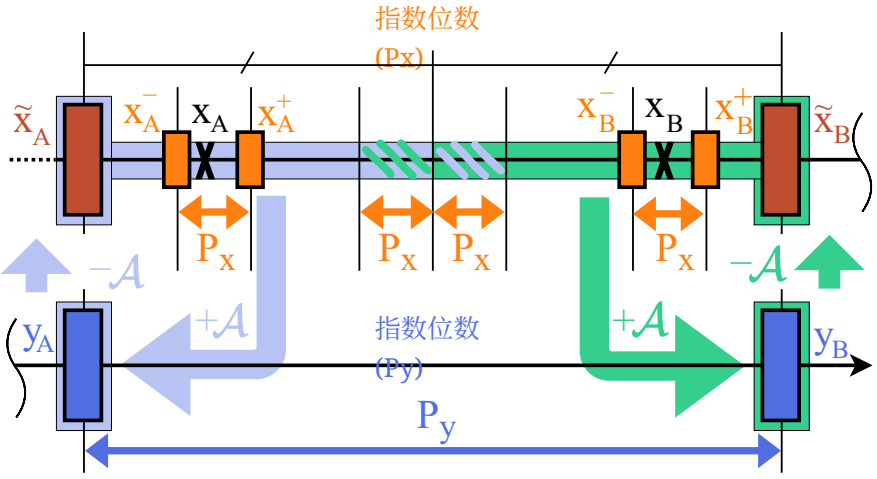

图5：紫色区域中的所有 $x_A \in \mathscr{D}$ 都会变换为 y_A，并用 $\tilde{x}_A$ 恢复。绿色区域中的所有 $x_B \in \mathscr{D}$ 都会变为 y_B 和 $\tilde{x}_B$。来自两种颜色重叠区域的样本，其结果取决于 $\mathcal{A}$ 和 P_y。

3.1.1 恢复误差

当 y_i 的精度低于数据集中原始 x_i 的精度时，就会产生恢复误差。当所有 y_i 具有相同的 E_U 时，它们将共享新的精度（见图 4）。 $\mathcal{A}$ 值越大，数字共享的公共位越多，恢复误差也越大。下面通过一个例子来说明。假设 $x_1 = 1.5$，且 $\mathcal{A} = 10^7$。那么，采用 IEEE 754 的舍入到最近值舍入方法后（还支持另外三种方法 [7]）， $y_1 = x_1 + \mathcal{A} = 10,\ 000,\ 002$，因为所用的 $\mathcal{A}$ 值使 y_1 的精度为 $P\ (E_U = 23) = 1.0$。解压缩后，恢复的样本将是 $\tilde{x}_1 = y_1 - \mathcal{A} = 2.0$，而不是 1。5，产生的绝对误差为 $\Delta_1 = 2.0 - 1.5 = 0.5$。

图5总结了从两个实数 $x_A,\ x_B \in \mathbb{R}$ 开始的变换及其逆变换，以及它们恢复后的形式 $\tilde{x}_A$ 和 $\tilde{x}_B$。考虑到 x_A，我们采用 IEEE-754 规则，用一个 32 位浮点数表示它，因此将其存储为可表示浮点数集合中最接近的邻近值之一 x_A^+， x_A^-。它们的精度为 P_x。经过变换后，我们得到 $y_A = x_A + \mathcal{A}$，其精度为 P_y。为了确保更多尾数位相同，我们选择 $\mathcal{A}$，使得 $P_y > P_x$。通过应用逆变换来恢复样本，我们得到 $\tilde{x}_A = y_A - \mathcal{A}$。 $\tilde{x}_A$ 和 $\tilde{x}_B$ 都属于 P_x 区域中的数字子集，其中没有小于 P_y 的分量，因为它们的尾数最低有效部分至少有 $\log_2(\frac{P_y}{P_x})$ 个零。假设所有样本变换后都具有相同的 $E_U\ (P_y)$，那么 $\tilde{x}_A$ 和 $\tilde{x}_B$ 将近似每个满足 $\tilde{x}_A \leq x_i \leq \tilde{x}_B$ 的 x_i。这是因为，给定 $x_i \in \mathscr{D}$，在加法变换 $y_i = x_i + \mathcal{A}$ 过程中，我们会丢失所有小于 P_y 的信息，而通过 $\tilde{x}_i = y_i - \mathcal{A}$，我们会用零填充丢失的尾数位。因此，

$$\begin{aligned} x_i + \mathcal{A} = y_A,\ \tilde{x}_A = y_A - \mathcal{A}\ \forall x_i \in [\tilde{x}_A; \tilde{x}_A + P_y/2 - P_x] \\ x_j + \mathcal{A} = y_B,\ \tilde{x}_B = y_B - \mathcal{A}\ \forall x_j \in [\tilde{x}_A + P_y/2 + P_x; \tilde{x}_B] \end{aligned} \tag{8}$$

这意味着，属于同一精度区域的不同数值会产生不同的恢复误差。根据标准， y_i和 $\tilde{x}_i$对于 $x_i \in]\tilde{x}_A + P_y/2 - P_x;\ \tilde{x}_A + P_y/2 + P_x[$取决于舍入方法和所选的 $\mathcal{A}$。使用加法变换可确保恢复误差有一个上界，该上界是 P_y的函数

$$\Delta \leq 2^{E_U^y - 23 - 1} = \frac{P_y}{2}, \tag{9}$$

其中， E_U^y 是变换后所有样本共有的无偏指数。

3.1.2 选择加法参数

我们选择 $\mathcal{A}$，使其满足以下条件：

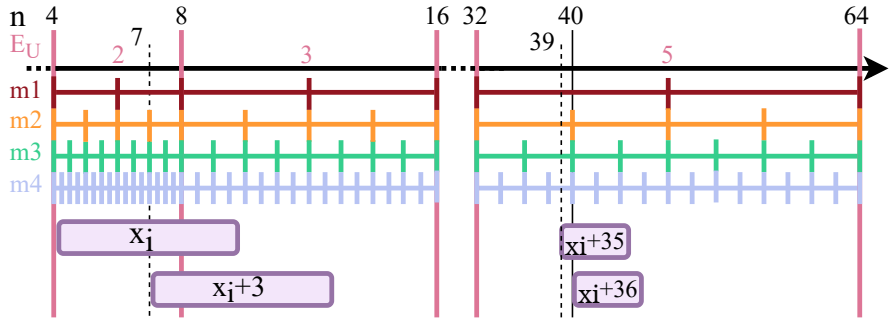

图6：当 紫色表示数据集取值范围时， $\mathcal{A}$ 应当被选中，使所有样本共享其指数。此外，在相同精度区域内，某些 $\mathcal{A}$ 比其他值产生更多公共位。

a. 在确保 Δ 和 δ 符合用户要求并满足 b) 和 c) 的前提下尽可能大；b. 变换后的数据集与该区域中最大的 2 的幂对齐；c. 其精度与和 $\mathcal{A}$ 相加所得数字的精度相同。

我们可以通过以下方式满足原则 [a.]：使用公式 (2) 和公式 (3) 中的误差界，或检查每个 $x_i \in \mathscr{D}$的实际误差。由于我们还希望所有 y_i 共享指数位，因此需要选择$\mathcal{A}$，使它们位于具有相同精度的区域。以图 6 中的示例为例，通过选择$\mathcal{A} = 3$，$E_U^{y_i} \in [2, 3]$；而选择 $\mathcal{A} = 35$时，$E_U^{y_i} = 5\ \forall y_i$。

原则 [b.] 源于比较图 6 中的选择 $\mathcal{A} = 35$ 和 $\mathcal{A} = 36$。尽管两者都会将数据集移至相同的指数区域，但 $\mathcal{A} = 35$ 会产生 $m_2 \in \{0, 1\}$；而选择 $\mathcal{A} = 36$时，$m_2 = 1$，从而使 m_2 共享。与所选区域内 2 的幂对齐是关键，这能最大限度地增加公共尾数位。与移位后的数据集对齐的 2 的幂越大，共享的尾数位就越多。

选择 $\mathcal{A}$时，需要遵循原则 [c.]，以避免出现反直觉现象。传统认知认为，$\mathcal{A}$选得越小，数据集中的变换后数值就越小、越精确，最终恢复误差也越小，但事实并非总是如此：只有在 $x_i + \mathcal{A} \in 2^{E_U^{\mathcal{A}}}$时才成立，其中 $E_U^{\mathcal{A}}$ 是 $\mathcal{A}$ 的无偏指数。选择符合这一最后要求的 $\mathcal{A}$，即可确保加法参数越小，恢复误差就始终越小。本文采用的 $\mathcal{A}$ 选择实现如下：

$$\text{Select } \mathcal{A} : \max(x_i \in \mathscr{D}) + \mathcal{A} = 2^{E_U^{\mathcal{A}}+1} - 2^{E_U^{\mathcal{A}}-23}. \tag{10}$$

3.2 乘法变换

该方法可概括如下：

$$y_i = \hat{x}_i \cdot \mathcal{M}, \quad \hat{x}_i = f(x_i, \mathcal{M}) \quad \forall x_i : x_i \in \mathscr{D} \tag{11}$$

其中，我们将 $\mathcal{M}$ 称为乘法参数；f 是一个函数，它会将数据集中的每个数据样本替换 为 $\hat{x}_i$，即与原始值任意接近、但具有一个优良性质的数值：与 $\mathcal{M}$相乘后会产生许多零。因此，与加法变换不同，我们对 $\hat{x}_i$ 而非原始 x_i应用乘法变换。我们的目标是让尾数中最低有效位为零的位数尽可能多。下面通过一个例子来说明。

3.2.1 数值示例

我们考虑 $x_1 = 363.754$和 $x_2 = 366$，其中 x_2 的尾数末尾有 16 个连续零位，而 x_1 则没有。为了增加末尾零的共同数量，我们建议将x_1 和 x_2 替换为经过精心选择的

乘法参数 $\mathcal{M}$相乘后能够呈现所需零位的数值。我们考虑 $\hat{x}_1 = 363.7894592285156$ 和 $\hat{x}_2 = 366.03509521484375$，其中 $\mathcal{M} = 57$。于是，$y_1 = \hat{x}_1 \cdot \mathcal{M} = 20736.0$和 $y_2 = \hat{x}_2 \cdot \mathcal{M} = 20864.0$共享末尾16个零。通过存储 y_1、y_2 和 $\mathcal{M}$，我们可以通过除法 $\tilde{x}_1 = y_1/\mathcal{M} = \hat{x}_1$和 $\tilde{x}_2 = y_2/\mathcal{M} = \hat{x}_2$ 还原原始数值，最大偏差为 $\delta < 0.01\%$。其他现有方法（如 [6], ）则直接近似 x_1，移除所有不需要的尾数最低有效位，使末尾达到 16 个零；还原后的样本为 $\tilde{x}_1 = 362.0$，偏差为 $\delta = 0.48\%$。

3.2.2 选择乘法参数

我们采用一种方法，将已知值（$\mathcal{M}$）映射到已知的尾数模式，使得它们相乘后得到全零序列。可以用十进制中的类似情况来理解这一点：我们知道，所有以.5结尾的数字乘以 $\mathcal{M} = 2$ 后，结果都以.0结尾；或者说，以 1.25×8 结尾的数字乘积总是以 0.00 结尾。浮点乘法也适用同样的概念：可以利用用二进制表示的模式，替换 x_i 的尾数。经过 $y_i = \hat{x}_i \cdot \mathcal{M}$ 后，从表示该模式起始 2 的幂次的位开始，y_i 尾数中的所有位都将变为零，因此零序列的长度可以任意选择。

表 2： $\mathcal{M}$ 二进制中的唯一模式（最高有效位为 1），以十六进制表示

$\mathcal{M}$模式	$\mathcal{M}$模式	$\mathcal{M}$模式	$\mathcal{M}$ 模式	$\mathcal{M}$ 模式	$\mathcal{M}$ 模式	$\mathcal{M}$ 模式	$\mathcal{M}$ 模式
3 0x2	11 0x2e8	19 0x35e50	27 0x25ed0	35 0xea0	43 0x2fa0	51 0xa0	59 0x22b63cbeea4e1a0
5 0xc	13 0x9d8	21 0x30	290x8d3dcb0	370xdd67c8a60	45 0xb60	530x9a90e7d95bc60	610x864b8a7de6d1d60
7 0x4	15 0x8	23 0x590	31 0x10	39 0xd20	470x572620	55 0x94f20	
9 0x38	17 0xf0	250xa3d70	33 0x3e0	41 0xc7ce0	49 0x14e5e0	57 0x23ee0	

表 2 列出了每个奇数 $\mathcal{M} \in \{3, \ldots, 61\}$对应的唯一模式。 $\mathcal{M}$ 越大，乘积的结果就越大，因此其精度越低，这通常会导致更大的恢复误差。该表还可以进一步扩展：针对每个 $\mathcal{M}$，纳入其因子的所有模式，这些模式也会产生零。我们不考虑偶数 $\mathcal{M}$，因为乘法中的 2 的幂只会影响指数，不会改变尾数。表 3 展示了这一过程的示例，其中

表 3：基模式替换和乘法示例

$x_i = 19.6$ $\mathcal{M} = 13$ **模式** = 0001 00111011 x_i **尾数** = 00111001100110011001101

$\hat{x}_i$	$E_U^{\hat{x}_i}$	Ext	$\hat{x}_i$ **尾数**	y_i	$E_U^{y_i}$	y_i **尾数**	**末尾零**	Δ
19.61538506	4		0011**1**00111011000 1001111	255.0	7	1111111**0**0000000000000000	16	0.02
19.38461494	4		00**1**10110001001 110110001	252.0	7	11111**0**000000000000000000	18	0.22
18.46153831	4		**0**0100111011000100111011	240.0	7	111**0**00000000000000000000	20	1.14
29.53846169	4	001	110110001 00111011000101	384.0	8	1**0**0000000000000000000000	22	9.94
19.69230843	4	0001	00111011 00010011101 1001	256.0	8	**0**0000000000000000000000	23	0.09

$x_i \to \hat{x}_i \to y_i = \hat{x}_i \cdot \mathcal{M}$。每个模式实例都用方框标出。带下划线的位与原始位不同，同一行中颜色相同的位表示相等的 2 的幂。扩展 = 尾数扩展：在寻找补丁时，我们可以在尾数后追加任意数量的 0，后面再追加一个 1。

$x_i = 19.6$ 会使用与 $\mathcal{M} = 13$ 相关联的模式，以多种方式进行替换。观察第一行，假设我们想通过替换 19.6 来获得 16 个末尾零，该数值需乘以 13 后才能求得，并且误差界要求 $E_U^{y_i} = 7$。因此，零将从 m_8 （红色）开始，它表示幂 $2^{7-8} = 2^{-1}$。 x_i 中对应同一幂的位是 m_5 （红色），我们应从该位置开始进行修补。我们使用模式的位移版本，以尽量减少对尾数（带下划线）的必要改动，从而限制误差Δ。此处需要将模式重复两次（见模式周围的方框），才能填满待修补的尾数部分。完成 m_{23} 后，我们通过舍入到最近值来截取它，得到 y_i。从这些示例中，我们可以得出结论：

- 模式具有位移不变性。
- 从左侧开始填补会导致更大的误差，并产生更多常见的零。
- 通常，修改原始尾数的位数越多，误差就越大。
- 尾数末尾有更多零并不一定意味着更大的恢复误差。

3.2.3 扩展到数据集

到目前为止，我们仅将乘法变换应用于最多包含 2 个数字的集合，并找到了许多性能各异的可能替代方案。然而，我们需要处理更大的数字集合。观察图1可知，我们希望找到一个适用于所有 x_i 的单一 $\mathcal{M}$，并使共有的末尾零数量最大化。在当前实现中，我们通过穷举搜索来处理这一问题：首先分析所有 $x_i \in \mathscr{D}$ $\forall \mathcal{M} \in [3, 61]$，寻找满足误差界的替代方案。在这些乘法参数中，我们选择使数据集的共有零数量最大的参数。随后，通过使用与所选 $\mathcal{M}$对应的替代方案，将每个 x_i 替换为$\hat{x}_i$，从而构建变换后的数据集；所选替代方案至少具有搜索过程中找到的最小末尾零数量，并且误差尽可能小。

3.2.4 乘法与校验

由于采用舍入到最近值近似方法，某些数字 $\tilde{y}_i$ 在通过 $\tilde{y}_i = \hat{x}_i \cdot \mathcal{M}$ 得到时无法具有$m_{23} = 0$。这些数字与末尾全为零的状态相差 P：乘法之后，我们可能需要用 $y_i = \tilde{y}_i \pm P$调整结果。

4 性能评估

为了评估我们提出的预处理技术的性能，我们使用表 4 中总结的设置比较压缩率。我们分析了所提出的加法在压缩率上的降低，以及

表 4：性能评估设置

压缩器	预处理	数据集集合	示例
Greedy-GD [9]	○ - 加法变换，使用 E^{Δ} 和 E^{δ}	aarhus citylab [10]	[35.87]
bzip2 [11]	▽ - 使用 E^{Δ} 和 E^{δ} 的乘法变换	chicago [12]	[20.5,0,-0.082,0.055,2,12.7]
lz4 [13]	× - 使用 E^{Δ} 和 E^{δ} [6]进行信息内容变换	cbb g2 [14]	[14310,403388]
Zstandard [15]	—— - 无损，通过以下方式移除小数部分	cbb 维2 [14]	[12856, 705226]
	将所有样本乘以 10 的幂	uci（*3500* 个样本） [16]	[3.5892]
		cmummac（*20000* 个样本） [17]	[-0.497314,0.425049,-0.036255,-0.755371]

乘法变换，与一种类似的有损预处理技术和一种无损方法进行比较。近期研究 [6] 还尝试增加尾数中的末尾零数量，以实现更好的压缩。该方法通过计算每个尾数位的信息含量，将低于某一阈值的末尾尾数位舍入为零。为了使该算法适用于我们的误差界方法，对于每个数据集中的所有维度，我们找出了满足所分析误差条件的最小保留信息含量阈值。无损预处理方法 则通过将所有样本乘以 10 的某次幂，直接去除数据集中的所有小数。

我们使用无预处理结果（$\mathrm{CR}_{\mathrm{NP}}$）作为性能比较的基线，公式为

$$\Delta \mathrm{CR}_{\%} = (\mathrm{CR} - \mathrm{CR}_{\mathrm{NP}})/\mathrm{CR}_{\mathrm{NP}} \cdot 100. \tag{12}$$

因此，$\Delta \mathrm{CR}_{\%} < 0$表示压缩效果有所提升。在图 7 中，横轴展示的是在施加误差界 $E^{\Delta} \in \{0.01, 0.1, 0.5, 1.0\}$ 和$E^{\delta} \in \{0.01\%\ 0.1\%\ 0.5\%\ 1.0\%\}$的情况下，恢复后数据集的实际最大误差。所选数据集涵盖了不同的数据类型、大小和维度数量组合：*aarhus citylab* [10]和 *uci* [16]用于一维浮点数，*cmummac* [17] 用于多维浮点数，*cbb gs* 和 *cbb dim2* [14]用于多维整数，*chicago* [12] 则包含整数和浮点数。我们使用的压缩器包括 *Greedy-GD* [9], ，这是一种采用重复数据删除且支持随机访问的基于字典的算法；此外，还选用了另外三种成熟的压缩器进行比较，即*bzip2* [11], 、*LZ4* [13] 和 *Zstandard* [15]。

图 7 的结果表明，除了在图 7c 中使用 E^{δ} 的情况外，对于所有数据集和压缩器，我们提出的预处理方法均优于 [6] 中的方法；在图 7f 中，压缩率最高提升了 46%。在图 7a 和图 7d 中，信息含量变换无法使误差低于预期界限，因此数据集未经处理，压缩率为 $\Delta \mathrm{CR}_{\%} = 0$。

对于所有数据集和压缩器，无论使用 $\Delta \leq 1$ 还是 $\delta \leq 1\%$，加法变换和乘法变换都能改善压缩效果，相较于未预处理数据集，降幅达

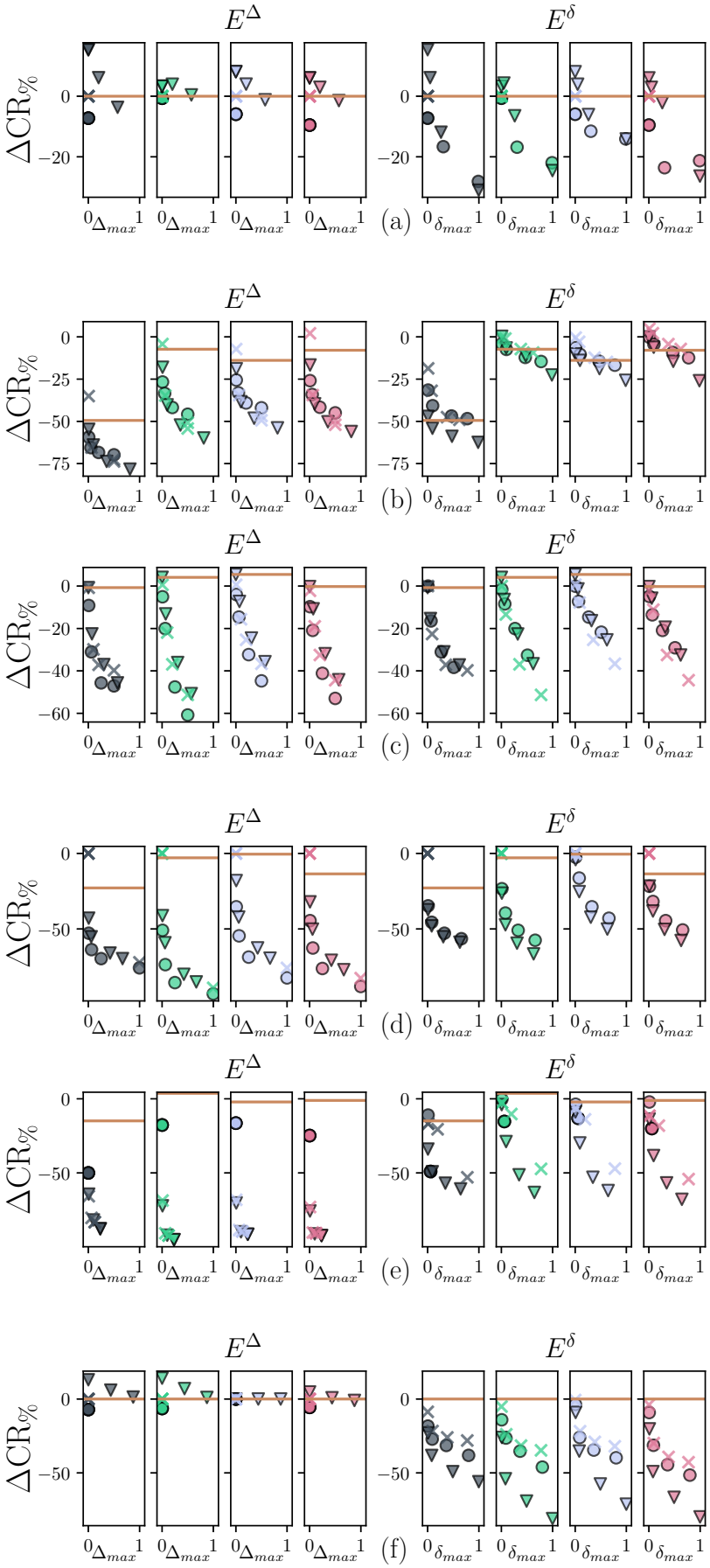

图 7：在一系列误差界、压缩器和数据集上，将新型预处理技术与 [6] (×) 中的有损方法以及无损方案（橙色线）进行比较的结果。所有结果均相对于未进行预处理时的压缩率（CR）计算，该值作为基线 $\Delta CR_{\%} = 0$。

最高可达 80%。我们可以在图 7a、图 7c 和图 7f 中看到一些情况：当 E^{Δ} 误差界较小时，未预处理数据集的表现更好；不过，通过在 E^{δ} 和 E^{Δ} 之间切换（反之亦然），或稍微放宽误差界，我们总能取得更好的效果。

我们还注意到，当加法变换和乘法变换的表现优于未预处理数据集时，它们也超过了无损压缩的表现（以橙色线表示）。在图 7a 和图 7f 中，这条线位于$y = 0$ 上，因为它们是整数数据集：无损算法对它们不起作用。正如预期的那样，可以通过接受更大的恢复误差来实现更高的压缩率。

此外，选择加法变换还是乘法变换，取决于数据集和误差界方法。例如，在图 7c 中，使用 E^{Δ} 时，加法变换优于乘法变换；而在图 7f 中，使用 E^{δ} 时，乘法变换效果更好。

关于误差界，选择 E^{Δ} 还是 E^{δ} 也取决于数据集。在图 7f 中，选择相对误差界显然比选择绝对误差界更有效；而在图 7b 中，绝对误差界的效果更好。一般来说，E^{Δ} 更适合数值接近零的数据集。选择 E^{Δ} 时应谨慎，以免损害数据集所承载的信息。图 7e 中有一个例子，E^{Δ} 可以将数据量减少近 95%；然而，正如表 4 中针对数据集 $cmummac$ 的示例所示，选择 $E^{\Delta} = 1$ 会使还原后的数据集无法使用。因此，在这种情况下，我们应选择 E^{δ}。

5 结论

本文提出了两种新颖的有损预处理技术：在压缩数据集之前对其进行变换，从而在给定误差界的条件下提高现有压缩器的压缩比。这两种方法利用加法和乘法等简单的浮点运算，以及浮点数据结构，来增加整个数据集中的公共位数量。我们通过比较这些技术得到的压缩率与未经预处理、经过无损预处理以及采用类似有损预处理技术 [6], 所得的压缩率，展示了其性能；比较涵盖四种压缩器和六个数据集。未来，我们计划研究将这些思路扩展到无损方法，并进一步缩短预处理时间。

参考文献

[1] G. Chiarot 和 C. Silvestri，《时间序列压缩：综述》，2021 年。 [在线]。获取地址：http://arxiv.org/abs/2101.08784[2] R. Vestergaard、D. E. Lucani 和 Q. Zhang，《一种可随机访问的时间序列数据无损压缩方案》，载于*IEEE INFOCOM* - 计算机通信会议，2020 年。[3] E. Latyshev，《用于预测性维护的传感器数据预处理、特征工程与设备剩余寿命预测》，载于 $DAMDID/RCDL$，2018 年。[4]*ISO/IEC 15948:2004* 信息技术 — 计算机图形与图像处理 — 便携式网络图形 *(PNG)*：功能规范，标准，2021 年。[5] I. Batal 和 M. Hauskrecht，《一种使用 DCT 和 DWT 的有监督时间序列特征提取技术》，载于国际机器学习与应用会议，2009 年。[6] J. D. e. a. M. Kl¨ower、M. Razinger，《将大气数据压缩至其真实信息含量》，$Nat\ Comput\ Sci$，2021 年。[7] J.-M. Muller、N. Brisebarre 和 F. e. a. Dinechin，浮点算术手册，2010 年。[8] *IEEE Std 754-2019*（修订 *IEEE Std 754-2008*）浮点算术 *IEEE* 标准，标准，2019 年。[9] A. Hurst、D. E. Lucani 和 Q. Zhang，《GreedyGD：面向物联网直接分析的增强型广义去重》，2022 年，已投稿至 IEEE 工业信息学汇刊。

[10] 奥胡斯市政当局，《传感器数据》，2017 年。 [在线]。获取地址：https://tinyurl.com/heeth2fd[11] J. Seward，《bzip2》，2019 年。 [在线]。获取地址：https://sourceware.org/bzip2/[12]芝加哥市，《海滩水质——自动传感器》，2022 年。 [在线]。获取地址：https://tinyurl.com/yz5yzy6a[13] Y. Collet、F. Handte、I. Rosen 和 R. Odaira，《LZ4——极快的压缩》，2011 年。 [在线]。获取地址：https://lz4.github.io/lz4/[14] P. Fr¨anti 和 S. Sieranoja，《K均值在六个聚类基准数据集上的性质》，《应用智能》，2018 年。[15]脸书，《Zstandard——实时数据压缩算法》，2015 年。 [在线]。获取地址：https://facebook.github.io/zstd/[16] H. Kaya 和 P. T¨ufekci，《燃气轮机一氧化碳和氮氧化物排放数据集数据集》，2019 年。 [在线]。获取地址：https://tinyurl.com/5bc7yx8u[17] J. M. e. a. F. De la Torre、J. Hodgins，《技术报告 cmu-ri-tr-08-22》，2009 年。 [在线]。获取地址：http://kitchen.cs.cmu.edu/index.php