FFCz：_科学数据频谱保持有损压缩的快速傅里叶校正

Congrong Ren俄亥俄州立大学俄亥俄州哥伦布 ren.452@osu.edu

Robert Underwood阿贡国家实验室伊利诺伊州莱蒙特 runderwood@anl.gov

Sheng Di阿贡国家实验室伊利诺伊州莱蒙特 sdi1@anl.gov

Emrecan Kutay俄亥俄州立大学俄亥俄州哥伦布 kutay.5@osu.edu

Zarija Luki´c Aylin Yener Franck Cappello Hanqi Guo劳伦斯伯克利国家实验室 俄亥俄州立大学 阿贡国家实验室 俄亥俄州立大学加利福尼亚州伯克利 俄亥俄州哥伦布 伊利诺伊州莱蒙特 zarija@lbl.gov yener@ece.osu.edu cappello@mcs.anl.gov

iversity 俄亥俄州哥伦布 o guo.2154@osu. edu

arXiv:2601.01596v1 [cs.DC] 4 Jan 2026

摘要——**本文介绍了一种新颖技术，该技术基于一种面向规则网格数据的新型快速傅里叶校正算法，在有损压缩中保留谱特征。对于宇宙学、湍流燃烧和X射线衍射等应用，同时保留数据的空间表示和频率表示至关重要，因为空间视图和频率视图能够提供互补的科学见解。具体而言，许多分析任务依赖频域表示来捕捉关键特征，包括宇宙学模拟的功率谱、燃烧中的湍流能谱，以及叠层成像中倒易空间的衍射图样。然而，现有压缩方法仅保证空间域中的精度，而忽视频域。为解决这一局限，我们提出一种算法，用于校正现成"基底"压缩器（如 SZ3、ZFP 和 SPERR）产生的误差，从而通过约束两个域中的误差，同时保留空间表示和频率表示。通过将频域误差表示为空间域误差的线性组合，我们推导出一个能够同时约束两个域误差的区域。给定基础压缩器产生的空间误差，以及用户定义的空间域和频域误差界，我们反复将空间误差向量投影到由空间和频率约束定义的区域，直至其落入二者的交集。我们还利用 GPU 并行性加速算法，以实现实用的性能。我们使用来自宇宙学模拟、X射线衍射、燃烧模拟和脑电图的数据集验证了该方法，结果表明，该方法能够有效保留空间域和频域中的关键科学信息。我们算法的开源 GPU 实现可在 https://github.com/rcrcarissa/FFCz 获取。**

索引词——**有损压缩、误差控制、傅里叶变换、科学模拟。**

I. 引言

误差有界有损压缩已成为现代科学数据分析中广泛使用的数据约简策略，用于应对数据生成速率与可用存储和通信带宽之间日益扩大的差距，从而保持性能和可扩展性。例如，Nyx 宇宙学代码仅运行五次模拟便会生成 2.8 PB 的数据，每次模拟包含 200 个快照 [1], ，这给存储系统和数据移动带来了巨大压力。不同于无损压缩（它能够在不损失信息的情况下减少数据量，但压缩比有限；对于浮点数据而言约为 2:1 [2]），误差有界有损压缩在保持数据质量的同时可实现显著更高的压缩比，并已成功应用于广泛的科学领域，包括宇宙学 [3], 、流体动力学 [4],和医学研究 [5] ；同时也被广泛用于降低存储成本 [6], [7], 、提升 I/O 性能 [8], 以及促进大规模数据分析 [9], [10]。

现有误差有界有损压缩算法的一项关键局限是，它们主要用于约束原始域（通常为空间域或时间域；为简洁起见，本文统称为空间域）中的重建误差，却忽略了频域中的失真，而频域对于许多数据驱动应用至关重要。例如，宇宙学数据的功率谱对于分析不同空间尺度上的物质与能量分布至关重要；如图 1 所示，如果只约束空间误差，却未能准确重建频率分量，功率谱就可能严重失真。湍流模拟数据的两点相关分析依赖于保留不同尺度上的能量，而频域中的误差会扭曲能谱估计 [11]。对于脑电图数据，未能保留频域信息可能导致对神经节律的误读，并损害临床相关模式的检测 [12]。

尽管谱保真度至关重要，现有压缩器却缺乏直接且严谨的机制来控制频域精度。迄今为止，谱特征的保留一直采用间接且经验性的方法。例如，用户必须通过反复试错手动调整压缩参数（如空间误差界）

直到达到所需的频谱精度。这一过程不仅需要投入大量精力，通过经验确定可行的空间误差界，还迫使压缩器满足过于严格的空间界，从而显著降低可实现的压缩比。这些挑战凸显了开发新方法的必要性，以便在压缩比、双域精度以及大型数据集上的性能之间取得平衡。

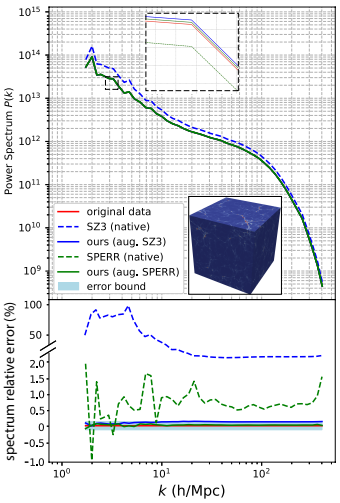


图 1. Nyx512^3 数据集的重子密度场功率谱。使用 SZ3 和 SPERR 进行压缩，并由我们的方法进行编辑，且比特率相同（即用于编码单个值的比特数），均为 0.023。空间域和频域相对误差界分别为 0.01% 和 0.1%。

我们提出了一种对偶域误差界范式，并为有损压缩提供了端到端的GPU实现，以便在空间域和频域中保持数据保真度，同时采用基于编辑的策略，适用于一维、二维、三维及更高维的数据集。据我们所知，这是首个能够使用户在两个对偶域中同时严格施加误差界的算法。我们的核心思想是将频域误差建模为空间域误差与复数系数的线性组合，从而刻画并限制满足两个域中的约束。具体而言，我们将空间域误差的可行域定义为两种几何结构的交集：由空间误差界导出的轴对齐（超）立方体（简称 s-立方体），以及由频域误差界限诱导的旋转（超）立方体（简称 f-立方体）。给定任意有损压缩器（即所谓的“基底”压缩器，例如 SZ3 [13]）生成且满足空间误差约束的压缩数据，我们首先找出误差超过频率误差界的频率分量，并将空间误差向量投影到 f-立方体上。接下来，我们定位违反空间误差约束的空间分量，并将误差向量投影到 s-立方体上。我们反复进行这些投影，直至误差向量落入两个立方体的交集，从而调整数据以满足两个域的要求。最终的编辑内容即为使空间误差向量进入可行域所需的调整。我们还会对这些编辑内容进行量化和压缩，以减少存储开销（如下文所示，存储编辑内容最多只需占压缩后存储空间的 10%）。为了进一步提高效率，我们利用 GPU 并行性，将点投影到立方体上，并验证各数据块是否满足约束。

我们通过全面比较，评估了我们的方法与最先进的误差有界有损压缩器。具体而言，我们使用 SZ3、ZFP [14],和 SPERR [15]作为基础压缩器。实验结果表明，在相同比特率下，我们的方法始终能够更好地保留频域特征，同时保持可比的空间域保真度，并且在宇宙学、叠层成像和燃烧等领域的各类数据集上均表现快速。此外，我们还评估了功率谱这一准确性指标，它对于表征许多物理模拟中的物质统计分布或涨落至关重要。我们的贡献如下：

- 一种新颖的算法，可保留多维数据集在空间域和频域中的信息，实现双域精度保证；
- 一种高效的 GPU 并行算法，可加速所提出的校正算法；
- 采用频域新指标，包括谱信号噪声比（SSNR）和功率谱相对误差；
- 在多种真实世界数据集和三种先进压缩器（SZ3、ZFP 和 SPERR）上进行了全面评估。

II. 相关工作

我们总结了误差有界有损压缩的相关文献，回顾傅里叶变换及其应用，并探讨其与谱域分析的联系。

A. 误差有界有损压缩

误差有界有损压缩器可确保每个重建数据点与其原始值之间的偏差不超过用户指定的界限，从而控制空间域中的误差，但无法保证频域中的偏差。例如，ZFP [16] 和 SPERR [15]等基于变换的方法会应用分块正交变换（例如整数到整数变换或类小波基），但这些变换是局部变换，而非谱变换，捕获的是邻近数据点之间的相关性，而非全局频率分量。SZ [2], [13], [17]–[19],等基于预测的压缩器会执行逐点预测和量化，确保每个数据点的偏差有界，但不会考虑跨频率相关性。因此，即使空间误差受到严格控制，其在频域中的累积效应仍可能没有上界，导致功率谱及其他频率相关分析出现失真，如评估部分所示。

B. 特征保持型有损压缩

研究人员一直致力于在有损压缩中保留感兴趣量（QoI），而不仅仅是空间域精度；这一方向将关注重点从逐点保真度转向支持后续分析的、具有科学意义的属性。这些感兴趣量包括拓扑、统计或物理不变量 [9]。例如，Liu 等人提出了 QPET [20], ，该方法保留数据的分位数分布，以维持关键的统计特征。

在这一方向上，压缩后增强或基于编辑的范式已成为施加 QoI约束的有效方法。这些方法针对基础压缩器的解压缩输出进行操作，通过对解压缩输出进行编辑来施加 QoI约束，同时不违反原始误差界。例如，Li 等人提出了 MSz，该方法针对二维和三维分段线性标量场，推导出能够保留名为 Morse–Smale分割的拓扑描述符的编辑 [21]。Gorski等人开发了一种框架，用于量化并应用调整，以保留等值线树 [22]。然而，保留谱特征仍是一个尚待解决的问题，后文将对此作进一步讨论。

C. 傅里叶与谱域分析

基于傅里叶变换的方法能够在大规模数据分析中高效地进行特征提取和模式识别。快速傅里叶变换 （FFT）是一种加速傅里叶变换计算的算法，也能加速卷积、滤波和相关等基础信号处理操作，这些操作常用于大规模数据环境中的特征提取 [23]–[25]。例如，在医学数据集中，傅里叶变换广泛用于将信号片段分类，以检测疾病并分析心理任务 [12]。在多媒体数据集中，它支持对音频中的语音、音乐和噪声进行分类 [26], , 对图像进行相似性搜索 [27], [28],, 以及在视频中检测运动目标 [29]。

功率谱是一个尤为重要的傅里叶派生量，它揭示了不同频率分量之间的功率分布 [30]。功率谱曲线的形状由波数 k 和功率谱P（k）表征，能为数据的底层结构提供丰富的信息。谱中的峰值对应占主导地位的周期分量，而平坦区域则表示功率均匀分布的白噪声。许多物理过程都遵循幂律 行为， P（k）$\propto k^{-\alpha}$，在双对数图中表现为一条直线，其中 α 衡量大尺度与小尺度涨落的相对强度 [31]。相比之下，指数衰减， P（k）$\propto e^{-k/k_0}$，表明高频内容受到强烈抑制，这通常见于光滑场 [32]。Jin等人进行了细粒度的率失真建模，以配置现有压缩器 [33], ；而我们的工作提出了一种专门用于保持傅里叶域保真度（包括功率谱精度）的新算法。

III. 背景

离散傅里叶变换。离散傅里叶变换(DFT) [34]将离散信号从空间域转换到频域。对于一个包含 N 个数据点（记为 $\{x_n\}_{n=0}^{N-1}$）的一维数据集，DFT 通过以下方式计算其频率分量 X_k

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i\frac{2\pi k}{N}n},\ k \in \{0,\ 1,\ ...,\ N-1\}, \tag{1}$$

对于一个维度为 N_0N_1 的二维数据集 $\{x_{m,n}\}$及其频率分量 $\{X_{u,v}\}$，DFT 可推广为

$$X_{u,v} = \sum_{m=0}^{N_0-1}\sum_{n=0}^{N_1-1} x_{m,n} e^{-i2\pi\left(\frac{um}{N_0}+\frac{vn}{N_1}\right)},$$
$$u \in \{0,\ 1,\ ...,\ N_0-1\},\ v \in \{0,\ 1,\ ...,\ N_1-1\}$$

高维离散傅里叶变换遵循相同的模式，只是每个维度增加了额外的指数项。

功率谱。功率谱（记作 $P(k)$）是分析信号的基本工具，可用于检测主导频率，并识别白噪声等模式；白噪声表现为平坦谱。给定一个包含数据值 $\{x_{m,n,p}\}$ 的离散三维数据集，我们首先对涨落进行归一化，以移除平均背景，具体为

$$x'_{m,n,p} = \frac{x_{m,n,p} - \bar{x}}{\bar{x}},$$

其中， $\bar{x}$ 是在 $\{x_{m,n,p}\}$ 上的平均值。此步骤确保功率谱测量的是相对结构和变化性，而不是场的绝对幅值。然后，我们将 $\{x'_{m,n,p}\}$ 的零频率分量移至中心，以清晰展现正频率与负频率之间的对称性 [35]。接下来，将空间函数转换到频域:

$$X'_{u,v,w} = \text{FFT}(x'_{m,n,p}).$$

随后，功率谱累加与中心距离相同的频率的幅值:

$$P(k) = \sum_{u^2+v^2+w^2=k^2} |X'_{u,v,w}|^2,$$

其中， $|\cdot|$ 表示取复数值的幅值。

凸集投影（POCS）方法。我们将误差向量移入空间约束与频率约束交集的算法，是对凸集投影（POCS）方法的一种专门改编。POCS 是一种成熟的迭代技术，用于寻找两个闭凸形状交集中的点。如图 2 (a) 所示，给定两个（闭）形状 X 和 Y，该过程从任意初始点 p开始，并反复交替将其投影到各个集合上。当交集 $X \cap Y$ 非空时，可以保证迭代点收敛到一个点$p^* \in X \cap Y$。相反，如果 X 和 Y 不相交，迭代点便无法收敛到同一个点，而会在分别位于两个集合上的点 x^* 和 y^* 之间振荡（图 2 (b)）。在寻找两个超立方体交集中的点时，交替投影算法会以线性（几何）速率收敛到其交集中的某个点。收敛速率取决于超立方体的相对方向以及交集处活跃面的集合。当交集近乎相切时，收敛较慢；反之，横截交集会带来更快的收敛。

尽管在我们的工作流程中也可以采用其他方法投影到凸集的交集上，但我们选择了POCS方法，因为它简单、内存效率高，且适合在GPU上并行实现。方法

例如Dykstra算法和交替方向乘子法（ADMM）通常收敛得更快，但存储校正项或原始—对偶变量会带来更高的内存开销。类似地，Douglas–Rachford分裂可能收敛很快；然而，其迭代点可能收敛到两个集合交集之外的点。

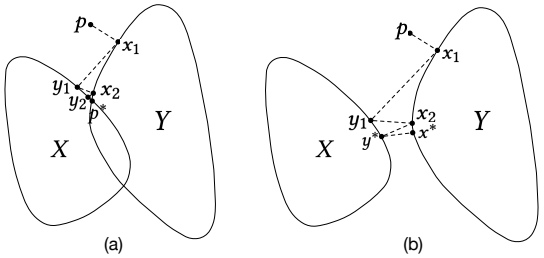


图2. POCS方法。该方法在两个闭凸集之间来回迭代投影任意点，直至其到达两个集合的交集中的某一点。只要两个集合的交集非空，就能保证收敛。

IV. 方法

本节首先 formulates 在空间域和频域（即所谓的“双域”）中界定数据误差的问题，然后以一维数据为例描述我们的算法，将其推广到多维数据，并介绍其并行 GPU 实现。

A. 双域误差界定问题

双域误差界定问题旨在界定重建数据在空间域和频域中的误差。给定原始数据 $\{x_n\}_{n=0}^{N-1}$ 和基础有损压缩器得到的解压缩数据 $\{\hat{x}_n\}_{n=0}^{N-1}$，我们将第 n 个空间域误差表示为 $\epsilon_n = \hat{x}_n - x_n$，将第 $\hat{}k$ 个频域误差表示为 $\delta_k = X_k - X_k$，其中 X_k 和 $\hat{X}_k$ 分别是对 $\{x_n\}_{n=0}^{N-1}$ 和 $\{\hat{x}_n\}_{n=0}^{N-1}$ 应用 FFT 后得到的第 k 个频率分量。在不失一般性的情况下，我们使用用户指定的全局空间误差界 E 和频域误差界 Δ 来描述我们的算法：[1]

$$\begin{aligned} &|\epsilon_n| \le E, \\ &|\Re(\delta_k)| \le \Delta \text{ and } |\Im(\delta_k)| \le \Delta, \end{aligned} \tag{2}$$

对于所有属于 $\{0,\ 1,\ ...,\ N-1\}$ 的 n 和 k，其中 $|\cdot|$、$\Re(\cdot)$ 和 $\Im(\cdot)$ 分别表示取绝对值、实部和虚部。

将频域约束映射到空间域。根据离散傅里叶变换的定义（式（1））， 频域误差 δ_k 是所有空间域误差 ϵ_n 的线性组合（采用复权重）：

$$\delta_k = \hat{X}_k - X_k = \sum_{n=0}^{N-1} \epsilon_n \exp\left(-i\frac{2\pi k}{N}n\right),\ k \in \{0,\ 1,\ ...,\ N-1\}, \tag{3}$$

[1]我们将在后续章节中讨论将误差界推广为逐点形式 E_n、 $\Delta_k^{\Re}$和 $\Delta_k^{\Im}$。

因此，可以将包含空间误差 ϵ_n 和频率误差 δ_k 的式（2）中的约束，表示为以下仅包含空间误差 ϵ_n的形式：

$$-\Delta \le \sum_{n=0}^{N-1} \cos\left(\frac{2\pi k}{N}n\right) \cdot \epsilon_n \le \Delta, \tag{4a}$$

$$-\Delta \le \sum_{n=0}^{N-1} \sin\left(\frac{2\pi k}{N}n\right) \cdot \epsilon_n \le \Delta, \tag{4b}$$

$$-E \le \epsilon_n \le E, \tag{4c}$$

对于所有属于 $\{0,\ 1,\ ...,\ N-1\}$ 的 n 和 k。

***f*-立方体和 *s*-立方体**。我们将式（4a）和（4b）中限制频域误差的约束称为 f-立方体，将式（4c）中限制空间域误差的约束称为 s-立方体；这些名称源于空间域误差向量空间中所呈现的几何形状空间误差向量 $\boldsymbol{\epsilon} = \{\epsilon_0, \epsilon_1, ..., \epsilon_{N-1}\} \in \mathbb{R}^N$（图 3 (a)）。 s-立方体和 f-立方体的维数均等于数据点的数量。为方便起见，对于与频率分量 k 相关联的某个向量$\mathbf{a}_k \in \mathbb{R}^N$，我们将形如 $\mathbf{a}_k^{\intercal}\boldsymbol{\epsilon} \le \Delta$ 的关于 $\boldsymbol{\epsilon}$ 的任意约束称为半空间约束半空间约束。形如 $-\Delta \le \mathbf{a}_k^{\intercal}\boldsymbol{\epsilon} \le \Delta$ 的约束则是一对半空间约束一对半空间约束。$\mathbf{a}_k$ 的第 n 个元素，记作 $a_{k,n}$，当频率分量 k 取实部时为 $a_{k,n} = \cos(\frac{2\pi k}{N}n)$，当取虚部时为 $a_{k,n} = \sin(\frac{2\pi k}{N}n)$。因此， f-立方体是针对一组频率分量的这类半空间约束对的交集。

频率基与空间基。考虑由余弦向量和正弦向量 $\{\mathbf{a}_k\}$ 构成的集合$2N$，它定义了 $k \in \{0, 1, ..., N-1\}$的半空间约束的法向量。根据厄米对称性 [36]，即 $X_{N-k} = X_k^*$，其中 X^*表示复共轭，只有频率索引 $k \le N/2$会产生不同的法向量。此外，当 $k = 0$ 或 $k = N/2$时，正弦项消失，只剩下余弦分量。总而言之，$2N$ 个向量 $\{\mathbf{a}_k\}$ 只产生 N个不同的向量。这些 N 个不同的向量在欧几里得内积下两两正交 stein2011fourier，构成 $\mathbb{R}^N$的一组替代正交归一频率基，区别于由 $\boldsymbol{\epsilon}$定义的标准空间基。在这种表示中， f-立方体在频率基中与坐标轴对齐（在空间基中旋转），而 s-立方体在空间基中与坐标轴对齐（在频率基中旋转），如图 3(a) 所示。

B. 交替投影校正算法

图 4 和算法 1 分别展示了我们的交替投影-校正算法的工作流程和伪代码。给定基础压缩器产生的原始数据 $\{x_n\}_{n=0}^{N-1}$ 和解压数据 $\{\hat{x}_n\}_{n=0}^{N-1}$，该算法迭代地将重建误差投影到 f 和 s 立方体上，并记录误差向量的位移：沿空间基的空间编辑以及沿频率基的频率编辑。

采用交替投影 *f*-立方体和 *s*-立方体。从解压数据的空间误差向量 $\boldsymbol{\epsilon}$ 开始，该向量最初位于 s-立方体内。我们采用类似 POCS 的方法，将其迭代投影到 f-立方体和 s-立方体上，直到

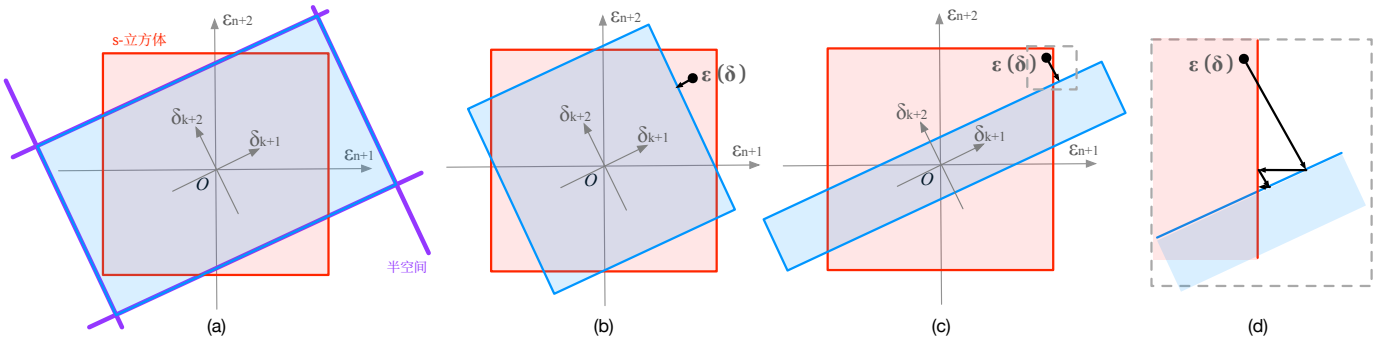


图 3. 我们方法的示意图。(a) 空间误差向量 ϵ的可行域，其中 ϵ_n 和 δ_k 分别表示空间基底和频率基底。红色和蓝色矩形分别表示 s-立方体和 f-立方体约束。沿 δ_{k+2} 方向的半空间对是冗余的，而沿 δ_{k+1} 方向的半空间对不是。(b) 位于s-立方体内、但在 f-立方体外的空间误差向量，通过裁剪其频率误差向量 $\boldsymbol{\delta}$，投影到 f-立方体上。如果它仍处于 s-立方体内，该过程便停止。(c) 否则，它将在 s-立方体和 f-立方体之间迭代投影，直到到达二者的交集。(d) 交替投影的详细示意图。

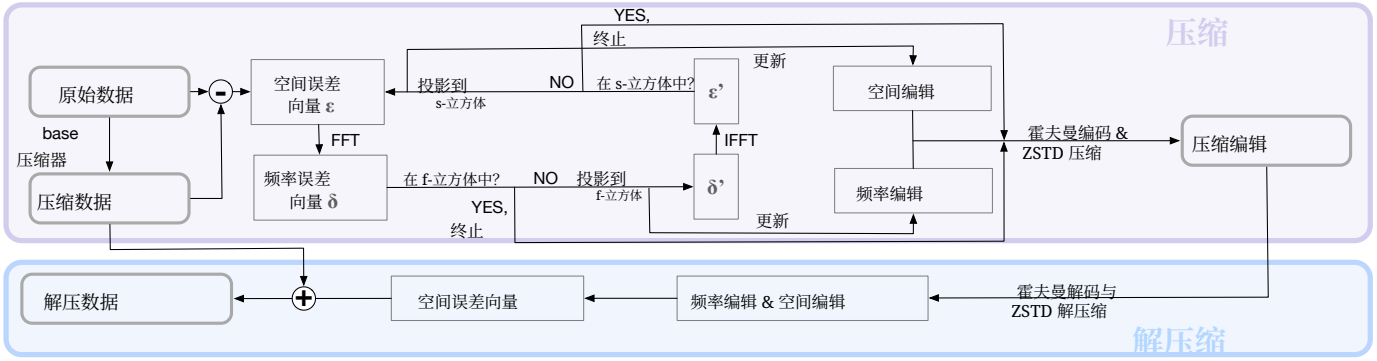


图 4. 我们交替投影的流程　　校正算法。我们迭代地将误差向量投影到 f 和 s 超立方体上，直到其收敛到它们的一个交　　r
集。

算法 1： 交替投影-校正

数据： 原始数据 $\mathbf{x}$，基础压缩器的解压数据 $\hat{\mathbf{x}}$；
E：用户定义的空间误差界；
Δ：用户定义的频域误差界

$\boldsymbol{\epsilon} \leftarrow \hat{\mathbf{x}} - \mathbf{x}$;
空间编辑 $\leftarrow$ []; /* 空间编辑 */
频域编辑 $\leftarrow$ []; /* 频率编辑 */
while 真 **do**
  $\boldsymbol{\delta} \leftarrow$快速傅里叶变换($\boldsymbol{\epsilon}$);
  如果*CheckConvergence*($\boldsymbol{\delta}$,Δ)**那么**
    跳出
  $\boldsymbol{\delta}' \leftarrow$ ProjectOntoFCube($\boldsymbol{\delta}$, Δ);
  频域编辑 $+= \boldsymbol{\delta}' - \boldsymbol{\delta}$;
  $\boldsymbol{\delta} \leftarrow \boldsymbol{\delta}'$;
  $\boldsymbol{\epsilon} \leftarrow$ 逆快速傅里叶变换($\boldsymbol{\delta}$);
  $\boldsymbol{\epsilon}' \leftarrow$ProjectOntoSCube($\boldsymbol{\epsilon}$, E);
  空间域编辑 $+= \boldsymbol{\epsilon}' - \boldsymbol{\epsilon}$;
  $\boldsymbol{\epsilon} \leftarrow \boldsymbol{\epsilon}'$;
空间域标志，紧凑空间域编辑 $\leftarrow$ CompactEdits(空间域编辑);
频域标志，紧凑频域编辑 $\leftarrow$ CompactEdits(频域编辑);
紧凑量化空间域编辑 $\leftarrow$ QuantizeEdits(紧凑空间域编辑);
紧凑量化频域编辑 $\leftarrow$ QuantizeEdits(紧凑频域编辑);
已压缩的空间域编辑 $\leftarrow$
  LosslesslyCompressEdits(紧凑量化空间域编辑);
已压缩的频域编辑 $\leftarrow$
  LosslesslyCompressEdits(紧凑量化频域编辑);
返回 空间域标志、频域标志、已压缩的空间域编辑、已压缩的频域编辑

它到达由f-立方体和 s-立方体的交集定义的可行域。我们首先得到频率误差向量 $\boldsymbol{\delta} = \{\delta_0, \delta_1, ..., \delta_{N-1}\}$ 对 $\boldsymbol{\epsilon}$执行快速傅里叶变换（如公式 (3) 所示），这等价于从空间基到频率基的坐标变换。通过将 $\boldsymbol{\delta}$ 中的所有元素与 $\pm\Delta$ 比较，可以检测是否违反了半空间约束。如果均未违反，则当前的 $\boldsymbol{\epsilon}$ 已满足 f 立方体约束，算法随即终止。否则，我们将 $\boldsymbol{\epsilon}$ 投影到 f 立方体上，具体做法是裁剪 $\boldsymbol{\delta}$，使其满足界限 $\pm\Delta$。接下来，对更新后的 $\boldsymbol{\delta}$ 执行逆快速傅里叶变换（IFFT），将其变换回空间基，得到更新后的 $\boldsymbol{\epsilon}$；然后通过裁剪超出界限 $\pm E$ 的坐标，将更新后的$\boldsymbol{\epsilon}$ 投影到 s 立方体上。交替投影到 f 立方体和 s 立方体的过程会不断重复，直到 $\boldsymbol{\epsilon}$ 同时满足这两项约束（图 3 (d)）。请注意，$\boldsymbol{\epsilon}$ 和 $\boldsymbol{\delta}$ 表示的是同一个误差向量，只是处于不同的基底中。

编辑量的紧凑化、量化与无损压缩。在交替投影过程中，我们分别将误差向量 $\boldsymbol{\epsilon}$ 沿空间基底和频率基底的偏移存储为空间编辑和频率编辑。随后将这些序列转换为紧凑形式，以减少基础压缩器之外引入的存储开销。投影结束后，每个编辑序列会进一步分解为两个向量：标志和紧凑编辑量。标志是长度为 N 的二进制向量，用于指示相应的空间或

频率分量是否为非零值；这些标志会打包为 8 位整数，以实现紧凑存储。紧凑编辑量仅存储非零项，其数量通常远少于 N。紧凑编辑量通过将 s 立方体或 f 立方体的每个轴划分为 2^m 个区间进行量化，其中 m 表示以位为单位的量化码长。我们在所有实验中固定 $m = 16$，以提供足够的精度，同时不增加存储开销。为确保量化后的编辑量仍能将 $\boldsymbol{\epsilon}$ 移入可行域，我们将初始误差界 E 和 Δ 分别略微收缩至 E（$1-2^{-m}$）和 Δ（$1-2^{-m}$）。空间编辑和频率编辑分开存储，因为频率编辑经逆变换后会影响所有空间编辑，产生一组稠密的非零项，因而导致内存使用效率低下。最后，使用霍夫曼编码 [37] 对标志和量化后的紧凑编辑量进行压缩，再使用 ZSTD [38] 进一步降低存储需求。

对解压数据应用编辑。要重构解压数据，我们首先进行 ZSTD 解压缩，然后进行霍夫曼解码，以获取频域和空间域中的标志与量化编辑量。接下来，我们对这些编辑量进行反量化，并重构完整长度的空间编辑量和频率编辑量。空间域中的完整编辑量通过将空间编辑量与频率编辑量的逆快速傅里叶变换相加获得。相应地，频域中的完整编辑量通过将频率编辑量与空间编辑量的快速傅里叶变换相加来恢复。最终，将完整编辑量加到基础压缩器的输出上，即可得到解压数据。

C. 推广到二维和三维问题

我们的方法可直接推广到二维和三维数据集，而无需改变 s-立方体和 f-立方体的形式。对于二维数据，空间误差 $\{\epsilon_{m,n}\}$ 必须满足与频率界相对应的线性组合：

$$-\Delta \le \sum_{m=0}^{N_0-1}\sum_{n=0}^{N_1-1} \cos\left(2\pi\left(\frac{um}{N_0}+\frac{vn}{N_1}\right)\right)\cdot\epsilon_{m,n} \le \Delta \text{ and}$$

$$-\Delta \le \sum_{m=0}^{N_0-1}\sum_{n=0}^{N_1-1} \sin\left(2\pi\left(\frac{um}{N_0}+\frac{vn}{N_1}\right)\right)\cdot\epsilon_{m,n} \le \Delta$$

对于所有频率索引（u，v），以及对于所有（m，n）的空间界$-E \le \epsilon_{m,\,n} \le E$。与一维情况一样，这些约束构成了维度为 N_0N_1 的空间中一个 f-立方体（频域）与一个 s-立方体（空间域）的交集，而三维数据则以类似方式扩展。

D. GPU 并行性

我们的算法在空间数据点和频率分量上进行并行化。GPU 内存中存储着四类变量：空间误差向量、频率误差向量、空间编辑及其变体（例如空间标志以及紧凑或量化编辑量），以及频率编辑及其相应的变体。

使用交替投影 f- 和 s- 立方体（算法 1 第 5–14 行）。我们首先使用高度优化的 cuFFT 库 [39]（第 5 行）对空间误差向量执行正向快速傅里叶变换，以获得频率误差向量。一个 CUDA 内核，CheckConvergence 随后检查所有频率误差分量是否都位于 f-立方体内，并为每个频率分量分配一个线程（第 6 行）。由于 f-立方体的边界超平面两两正交，其投影操作可分解为沿频率基各轴进行的独立投影。如果整个频率误差向量已位于 f-立方体内，算法便以空间编辑和频率编辑均为零的状态终止（第 7 行）。否则，我们调用 ProjectOntoFCube 内核，将频率误差向量投影到 f-立方体上，方法是将每个频率误差分量截断至区间 $[-\Delta, \Delta]$（第 8 行）；然后根据沿频率基的位移更新频率编辑（第 9 行），并相应更新频率误差向量（第 10 行）。尽管这些步骤是在单个内核 ProjectOntoFCube 中实现的，我们仍在算法 1 中将它们分别列出，以阐明编辑和误差向量的更新方式。接下来，逆快速傅里叶变换（通过 cuFFT）将更新后的频率误差向量变换回空间域（第 11 行）；随后由 ProjectOntoSCube 内核将空间误差向量投影到 s-立方体上，并通过为每个空间误差分配一个线程来更新空间编辑和误差向量（第 12–14 行）。同样，投影到 s-立方体也可分解为沿空间基各轴进行的独立投影。

编辑的紧凑化、量化和无损压缩（算法 1 第 15–20 行）。如第四-B 节所述，为了高效表示编辑，我们将其拆分为两个部分：用于指示非零项位置的二进制标志，以及仅包含非零值的紧凑编辑向量。在 CompactEdits 内核中计算互斥前缀和（第 15–16 行），以确定非零编辑的输出位置。随后，使用 QuantizeEdits 内核（第 17–18 行）并行量化紧凑向量中的每个编辑，每个编辑分配一个线程。最后，采用霍夫曼编码对量化编辑量进行无损压缩。LosslesslyCompressEdits 内核首先将符号划分为数据块，并行按块计算符号出现次数，然后累加结果以构建全局频率表（第 19–20 行）。

V. 评估

本节介绍评估方案，并重点阐述关键观察结果。

A. 评估方案

基础压缩器。鉴于 SZ3 [13],ZFP [14], 和 SPERR [15] 应用广泛且涵盖多种算法设计，我们选取它们作为具有代表性的基础压缩器。它们是先进且应用广泛的有损压缩器，能够在各种科学应用中提供精细的逐点误差控制。

基线方法。在比较中，我们还使用 SZ3、ZFP 和 SPERR 作为基线方法，以便直接评估相较于未经修改的压缩器，我们的方法所带来的性能提升。

评价指标。我们通过定量指标比较不同方法，例如我们的方法为基础压缩器带来的压缩比（表 II）、率失真（图6）和吞吐量（图 7）。具体而言，我们使用谱信噪比（SSNR） [40], [41]评估频域中的精度：

$$SSNR(\hat{X}_k, X_k) = 10\log_{10}\left(\frac{\sum_k |X_k|^2}{\sum_k |X_k - \hat{X}_k|^2}\right).$$

SSNR 越高，频域失真越小。由于 Parseval 定理保证空间域与频域之间的能量守恒 [42]„ 这也意味着快速傅里叶变换（FFT）同样保持均方误差（MSE），因此我们不使用峰值信噪比（PSNR）衡量频域失真：

$$MSE(\hat{X}_k, X_k) = MSE(\hat{x}_n, x_n).$$

因此，PSNR 在两个域之间的差异仅在于归一化因子，而 SSNR 能更有意义地衡量频域中的精度。解压数据在空间域中的精度则通过 PSNR 衡量。我们将相对频率误差（RFE）定义为

$$\text{RFE of component } l = \frac{|\text{frequency error } l|}{\max_k |\text{frequency component } k|}.$$

我们还对这些压缩算法进行定性比较，包括功率谱（图10）和解压后空间数据的可视化结果（图5）。

表 I 基准数据集

数据集	dim	size	属性	精度
Nyx（高分辨率）	3D	$2,048^3$	重子密度，暗物质密度	单个
Nyx（中等分辨率）	3D	$1,024^3$		
Nyx（低分辨率）	3D	512^3		
S3D	3D	500^3	甲烷、氧气、一氧化碳、二氧化碳、水、氮气	双精度
HEDM	2D	$2,048^2$	归一化的	双精度
EEG	1D	31,000	标准	双精度

数据集。表 I 总结了评估中使用的数据集。这些数据集涵盖多个科学领域、空间和时间分辨率以及数据模态，体现了科学数据分析所面临的多样化挑战。Nyx数据由使用 Nyx代码进行的宇宙学模拟生成 [1]。S3D数据来自燃烧模拟，这些模拟能够解析小尺度湍流反应流的所有细节，包括流体动力学和详细化学反应 [43]。高能衍射显微镜（HEDM）数据记录了通过对晶体样品进行X射线成像获得的衍射射线强度，具有较高的时间分辨率。脑电图（EEG） [44] 是一种记录连续脑活动的医学时间序列数据库。

平台。我们在国家能源研究科学计算中心（NERSC）的 Perlmutter 超级计算机上进行实验 [45]。每项实验使用一块运行 CUDA 12.4 的英伟达 A100 图形处理器（40 GB HBM2内存），或一颗拥有 64 个核心的超威半导体 EPYC 7763中央处理器。

B. 关键观察结果

观察 1。与相应的基础压缩器相比，我们的方法引入的编辑仅导致压缩比小幅下降（SZ3 和 SPERR 约下降 15%，ZFP 下降 0.001%）。

表 II 报告了三种情况下的压缩数据大小：(1) 仅限制空间误差的基础压缩器（原生方式）；(2) 通过收紧用户指定的空间误差界，同时限制空间误差和频率误差的基础压缩器（试错法）；以及 (3) 我们的增强方法，对情况 (1) 的结果进行编辑，以额外限制频率误差。对于所有数据集，相对空间误差界固定为 ϵ(%) =0。1。RFE界的选取方式为：使用误差界为 $\epsilon(\%) = 0.1$ 的基础压缩器时，重建数据的最大频率误差会降低 100 倍。

我们的编辑对压缩比的影响很小。在所有数据集上，试错法都会显著降低压缩比，有时甚至会降低几个数量级。相比之下，我们的方法在保留接近原始压缩比的同时，在基础压缩器输出之上施加频域界。

我们的编辑效果因基础压缩器而异。对于 SZ3，我们的方法引入的编辑仅使压缩比降低约 10-20%，不会主导整体压缩后大小。对于 ZFP 和 SPERR，这种影响更小；对于 ZFP 甚至可以忽略不计（约为 10^{-5}）。这种差异源于：SZ3 作为一种基于预测的压缩器，在不利用全局相关性的情况下，根据局部邻点估计每个数据点，因此在保留频率内容方面效果较差，尤其是在高频范围内。相比之下，ZFP 和 SPERR 作为基于变换的方法，会利用更广空间范围内的相关性，从而天然保留更多频域结构。

如图5所示，我们的方法对压缩比影响较小，这可归因于空间域和频域中的有效编辑数量有限。在第三列中可以看到，非零（有效）编辑呈稀疏分布，这意味着大多数空间和频率分量保持不变。然而，当我们仅在一个域中将编辑前后的差异可视化时，该差异是通过对两个域中的编辑求和，并根据需要应用正变换或逆变换计算得到的，此时变化会出现在该域的所有分量中（见第四列）。这一观察也说明了我们为何要维护两组独立的编辑，分别对应两个域。

观察2。在相同的比特率（或存储大小）下，与基线方法相比，我们的方法能够实现更高的频域精度，同时保持相当的空间域精度。

我们在图6中绘制了所有方法的信号噪声比与比特率的关系，其中我们的方法应用于$\epsilon(\%) = 0.1$处基础压缩器的结果。相同比特率下，我们的方法始终能获得更高的信号噪声比。在空间

表 II 我们的方法与基础压缩器的压缩比。对于 Nyx 和脑电图数据集，$\epsilon(\%) = 0.1$ 和 $\delta(\%) = 0.1$；对于 S3D 数据集，$\epsilon(\%) = 0.1$ 和 $\delta(\%) = 10^{-5}$；对于 HEDM 数据集，$\epsilon(\%) = 0.1$ 和 $\delta(\%) = 10^{-7}$。

数据集	属性	压缩比								
		SZ3			ZFP			SPERR		
		ϵ 仅	ϵ 和 δ	我们的增强方法	ϵ 仅	ϵ 和 δ	我们的增强方法	ϵ 仅	ϵ 和 δ	我们的增强方法
Nyx (高分辨率)	重子密度	44,307.1	218.8	34,957.7	406.0	50.6	405.8	16,834.6	2,159.1	13,427.3
	暗物质密度	931.6	11.2	924.2	36.4	6.8	36.4	532.2	103.1	532.0
Nyx (中等分辨率)	重子密度	28,981.0	108.3	22,737.3	224.0	50.9	224.0	10,886.7	1,328.0	9,532.1
	暗物质密度	645.0	11.0	636.6	19.5	5.7	19.5	377.5	96.9	377.4
Nyx (低分辨率)	重子密度	9,946.0	71.4	7,231.0	148.0	51.3	148.0	3,907.2	661.1	3,126.9
	暗物质密度	334.9	11.3	327.8	12.4	5.7	12.4	210.4	102.9	210.4
S3D	CO2	2,822.6	100.9	2,514.3	95.1	35.4	95.1	1,864.0	700.3	1,864.4
HEDM	归一化	4,632.5	3,190.0	3,813.4	628.9	567.2	628.9	993.9	256.5	993.9
EEG	标准	13.7	11.5	13.7	1.8	1.2	1.8	-	-	-

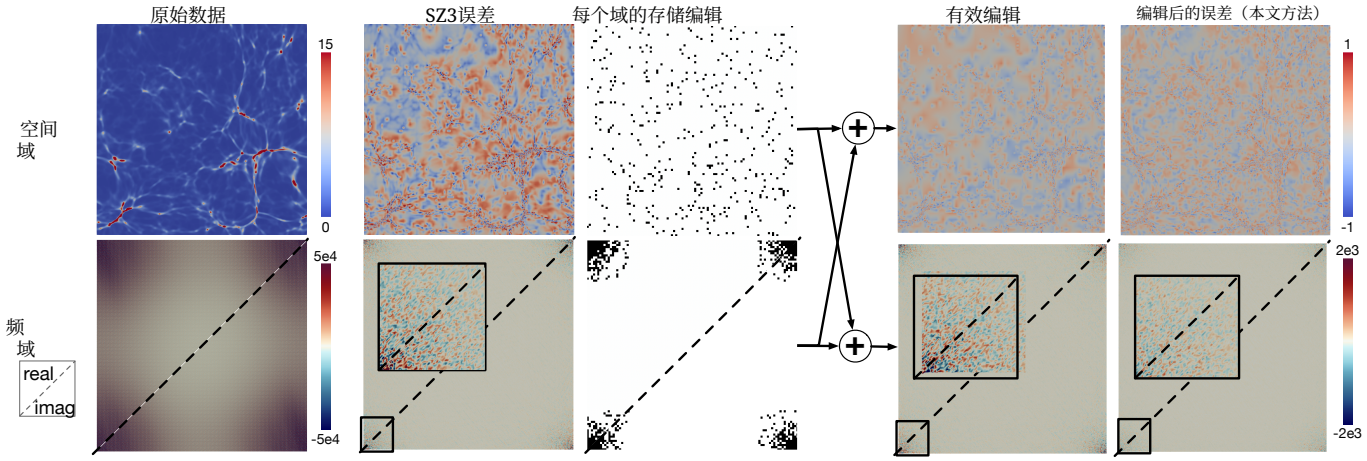


图5. 将我们的方法应用于低分辨率 Nyx数据中经 SZ3 压缩的重子密度场，在一个二维切片上的可视化结果，$\epsilon = 1$ 和 $\delta = 2,000$。第三列显示了我们的方法生成的空间编辑和频率编辑的位置，每种编辑都修改单一域中的分量。空间编辑和频率编辑分别有 412 个和 1706 个，分布稀疏。第四列所示的给定域中的编辑总数，是将两个特定于域的编辑相加得到的；必要时应用正向或逆快速傅里叶变换，将它们表示在同一域中。

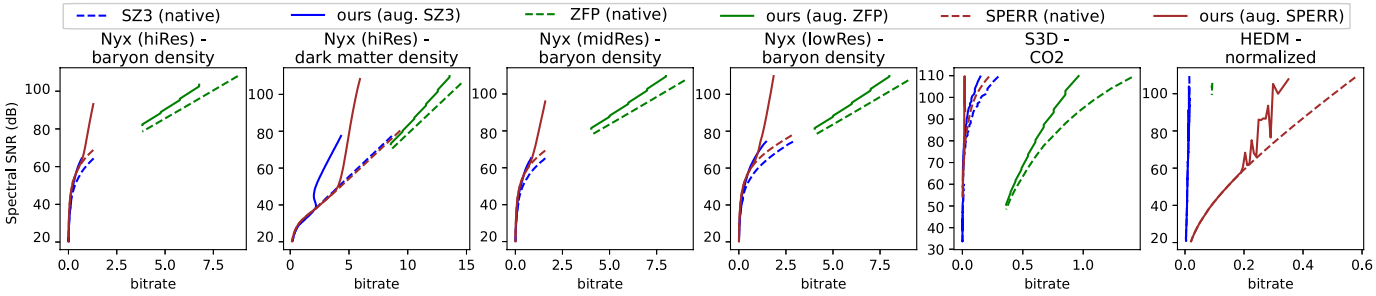


图6. 我们的方法、SZ3、ZFP 和 SPERR 在频域中的信号噪声比与比特率对比。应用我们的方法时，我们在基础压缩器的结果上进行编辑，其空间误差界为 ϵ（%）$= 0.1$。

域中（图8），我们观察到，为了保持空间精度，我们的方法不需要额外存储。尽管它会增加编辑数据的存储空间，但编辑过程会减少空间误差，因此曲线仅发生微小变化。

观察3。我们的编辑过程并非整体运行时间的瓶颈。

图7（a-c）展示了基础压缩器的吞吐量，以及应用于这些压缩器的编辑过程的吞吐量。编辑步骤通常快 2 至 5 倍，这表明它并非整体运行时间的瓶颈。唯一的例外是使用 ZFP 压缩 HEDM 数据集时，此时基础压缩器的吞吐量与我们的编辑过程相当。这是因为 HEDM 数据集中的非零值很少，而三种基础压缩器处理这类数据的机制各不相同

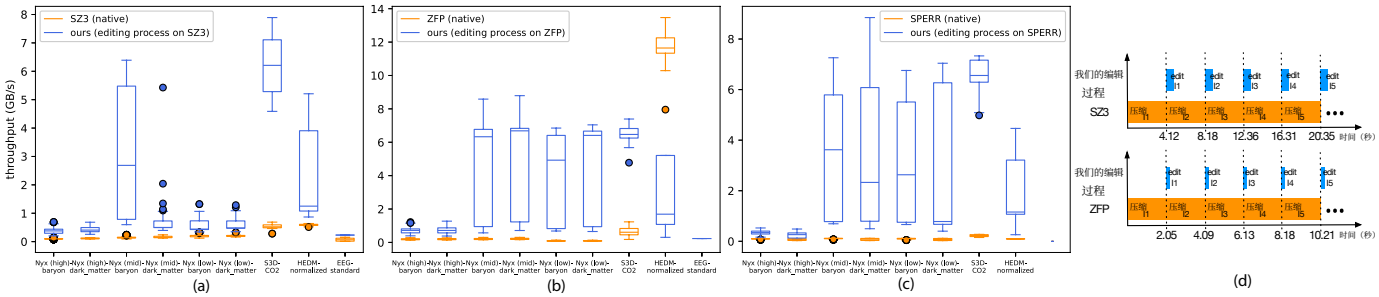


图7。（a-c）比较我们的方法、SZ3、ZFP 和 SPERR 在不同误差界下的平均吞吐量。我们的编辑过程不包括基础压缩器执行的压缩。（d）展示了使用 SZ3 和 ZFP，并通过我们的方法增强，对来自 Nyx（低分辨率）的多个实例执行流水线压缩-编辑工作流的时间线。与 SZ3 或 ZFP 的压缩过程相比，我们的编辑过程仅占用很小的时间比例，因此不会影响工作流的总体耗时。

。典型的速度排序是 ZFP 最快，其次是 SZ3，最后是 SPERR。这是因为 ZFP 针对全零块设有快速处理路径：它会对每个数据块快速执行范数检查，然后决定输出较短还是较长的编码。相比之下，SZ3 会预测并量化每个数据点，不会跳过数据块或数据点。另一方面，SPERR 会执行完整的多级变换并扫描整个数据集，因此通常比 SZ3 慢。

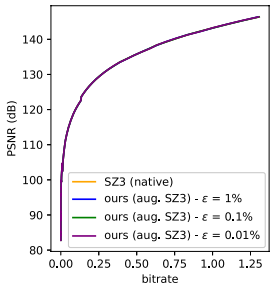


图8。Nyx（高分辨率）数据重子密度在空间域上的峰值信噪比与比特率对比，比较了我们的方法和 SZ3。我们在基础压缩器的结果上进行编辑，其空间误差界为 ϵ（%）$= 1$、0。1 和 0。01。

如表III所示，我们的编辑过程的运行时间与交替投影的迭代次数密切相关。然而，迭代次数并不一定与频域误差界成正比。相反，它更多取决于空间误差向量的初始位置以及 s 立方体和 f立方体的几何形状。较小的频域误差界（例如 10^{-4} %或 10^{-5}%）对应较小的f立方体，而该立方体通常被 s立方体包围。在这种情况下，投影通常在第一次迭代后便会终止，而不会引入

执行任何非零（活跃）空间编辑。相比之下，当s- 和 f-立方体部分重叠时，需要更多迭代次数，且收敛情况取决于它们投影边界之间的夹角。

表III 我们的编辑过程中对 SZ3 进行交替投影的迭代次数、有效编辑数和运行时间：Nyx（低分辨率）上的重子密度结果，ϵ(%) $= 0.1$。

δ(%)	# 迭代次数	# 有效空间编辑	# 有效频域编辑	时间（毫秒）
10^{-2}	53	445	2,325	628
10^{-3}	98	1770	323,789	946
10^{-4}	1	0	16,353,370	378
10^{-5}	1	0	53,950,920	396

图 7 (d) 显示，在流水线式压缩–编辑工作流中，我们的方法不会增加总体运行时间

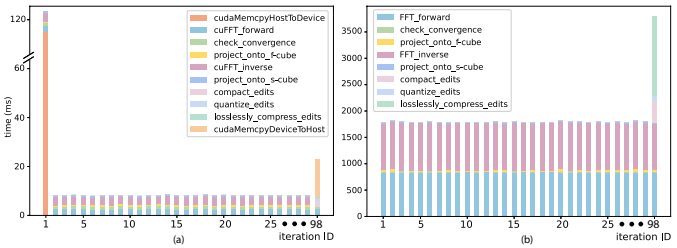


图 9. 在 Nyx（低分辨率）数据集中，对 SZ3 重建的重子密度应用我们的编辑过程时，(a) GPU 内核和内存传输以及 (b) CPU 函数的计时结果，其中 ϵ(%) $= 0.1$ and δ(%) $= 0.001$。由于每次迭代的计时变化较小，因此省略了若干次迭代。

针对一系列数据实例。当生成多个实例时（例如，使用不同参数或在不同时间步生成），压缩和编辑可以重叠进行：实例 i 的编辑与实例 $i+1$的压缩并行执行。因此，总运行时间与仅进行压缩的工作流程相同。

我们在内核/函数级和端到端级比较了校正算法的 GPU 与 CPU 实现，并将其应用于 Nyx（低分辨率）数据集中 SZ3 重建的重子密度，其中 ϵ(%) $= 0.1$ andδ(%) $= 0.001$（图 9）。选择该数据集是因为其 CPU 可扩展性良好，而所选误差界允许进行更多次迭代，从而更清楚地呈现每次迭代的计时变化。CPU 函数使用 OpenMP [46]进行并行化。所有计时均不包括初始 GPU 预热和数据加载开销。

在所有 GPU 内核中，快速傅里叶变换/逆快速傅里叶变换计算占主导地位，约占内核总执行时间的 68.7%。当算法在大约 10 次迭代后终止时，会表现出内存受限行为，因为内存传输占用了大部分运行时间。对于迭代 98 次的情况，算法则变为计算受限，此时内核执行占 GPU 总时间的 85.1 %，内存传输仅占 14.9%。

表 IV 报告了在相同参数下编辑 SZ3 结果时 GPU 内核和 CPU 函数的性能。指标包括执行时间（毫秒）、有效内存带宽（GB/s）及其相对于硬件峰值的效率、浮点性能（每秒十亿次浮点运算）和浮点运算效率，以及算术强度（AI，即每访问一个字节所执行的浮点运算次数）。对于

表 IV 编辑 SZ3 的结果时 GPU 内核和 CPU 函数的性能 ϵ(%) $= 0.1$ 和 δ(%) $= 0.001$。指标包括执行时间、有效内存带宽（BW）、相对于硬件峰值的带宽效率（BW Eff）、浮点性能（每秒十亿次浮点运算）、浮点运算效率（FLOP Eff）和算术强度（AI）。

内核/函数	平台	时间（毫秒）	带宽（GB/s）	带宽效率（%）	每秒十亿次浮点运算	浮点运算效率（%）	AI	加速比
正向快速傅里叶变换（cuFFT / FFTW）	GPU	2.53	423	26.4	7160	36.7	16.9	321
	CPU	813	1.32	0.66	22.3	0.29	16.9	
逆快速傅里叶变换 (cuFFT / FFTW)	GPU	3.34	321	20.0	5420	27.8	16.9	272
	CPU	910	1.18	0.59	19.9	0.26	16.9	
CheckConvergence	GPU	1.16	462	29	57.8	0.3	0.25	14.7
	CPU	17	31.6	16	7.89	0.1	0.25	
ProjectOntoFCube	GPU	0.65	1070	67	2000	10.3	1.25	26.2
	CPU	17	127.4	64	159.2	2.0	1.25	
ProjectOntoSCube	GPU	0.59	890	56	900	4.6	0.5	28.8
	CPU	17	31.6	16	7.89	0.1	0.25	
CompactEdits	GPU	2.56	628	39	52	0.27	0.08	27.3
	CPU	70	23	11.5	1.92	0.025	0.08	
QuantizeEdits	GPU	0.77	208	13	696	3.6	0.33	19.5
	CPU	15	107	53	35.7	0.46	0.33	
LosslesslyCompressEdits	GPU	0.81	665	42	33	0.17	0.05	67.9
	CPU	55	9.8	4.9	0.49	0.006	0.05	
cuda 主机到设备内存复制	GPU	115	4.65	0.29	0	0	0	-
cudaMemcpyDeviceToHost	GPU	15.2	35.3	2.2	0	0	0	-
端到端	GPU	**946**	**1100**	**69**	**4083**	**21**	**2.32**	**184**
	CPU	**174092**	**2.13**	**1.1**	**23.0**	**0.3**	**10.8**	

GPU 内核，其峰值性能为 19.5 TFLOPS（单精度浮点），第二代高带宽内存带宽为 1.6 TB/s；CPU 的单精度浮点峰值为 7.8 TFLOPS，内存带宽为 200 GB/s。

GPU实现相比CPU基线带来了显著加速，端到端加速达到184倍，总执行时间从174秒缩短至不到1秒。单个内核/函数的加速比介于14.7倍至321倍之间。对算术强度和效率指标的分析表明，快速傅里叶变换内核具有中等算术强度（约 17 FLOPs/字节）和较高的浮点运算效率（约30–37%），体现了计算受限与内存受限行为并存的特征。相比之下，大多数数据处理内核（如CompactEdits和 LosslesslyCompressEdits）的算术强度极低（$<$ 1FLOP/字节），浮点运算效率也很低，表明它们主要受内存限制。

观察4。我们的方法在相同比特率下能更好地保留功率谱。

我们的方法通过为各个频率分量分配不同的逐点相对误差界，能够保留功率谱。图10上排展示了我们的方法增强 SZ3 后重建得到的功率谱，其中每个功率谱分量的相对误差界均设为 0.1%。作为对比，图中还展示了比特率相同的 SZ3 结果。下排绘出了功率谱的相对误差 $\frac{\hat{P}(k)-P(k)}{P(k)}$，其中，$\hat{P}(k)$ 是由我们的方法或 SZ3 得到的重建功率谱，而 $P(k)$ 是原始功率谱。显然，我们的方法始终能将重建谱控制在指定误差界以内，而 SZ3 偶尔会超出该界。

六、结论与未来工作

我们介绍了 FFCz，这是一种适用于任意基础压缩器的图形处理器加速校正算法，可确保数据在空间域和频域中的保真度，弥补现有方法仅保留空间信息的不足。通过将频域约束转换到空间域，FFCz 将可行域表示为 s-立方体与 f-立方体的交集，并通过迭代交替投影来强制满足这些约束。该方法可与 SZ3、ZFP 和 SPERR 等压缩器无缝集成，以较小的存储和运行时开销确保其输出满足用户指定的双域误差界。我们在宇宙学、X射线衍射和燃烧领域数据集上开展的实验表明，所提方法能够保留下游分析所需的关键特征，从而支持可靠的双域数据解读。

尽管表现出色，仍有若干局限性值得在未来工作中加以探索。首先，迭代投影算法可能需要很多次迭代才能收敛，尤其是在基础压缩器产生空间不规则误差时。开发直接投影方案或混合投影方案，或许能提高收敛可预测性。其次，我们的方法需要用户指定空间与频率误差界，而这两者本质上相互耦合。采用自动化或数据驱动方法联合调节这些误差界，或许能提升易用性和效率。第三，将我们的算法事后集成到现有压缩器中，会带来一定的计算和内存开销。若将可行性约束直接嵌入压缩流程，或可实现更快、更紧凑的方案。最后，尽管本文展示的是带有傅里叶域约束的应用，该方法也可自然扩展到其他变换（例如小波或球谐函数），并可推广到多个域——空间、频谱、时间或多分辨率——以拓宽其在各类科学工作流程中的适用范围。

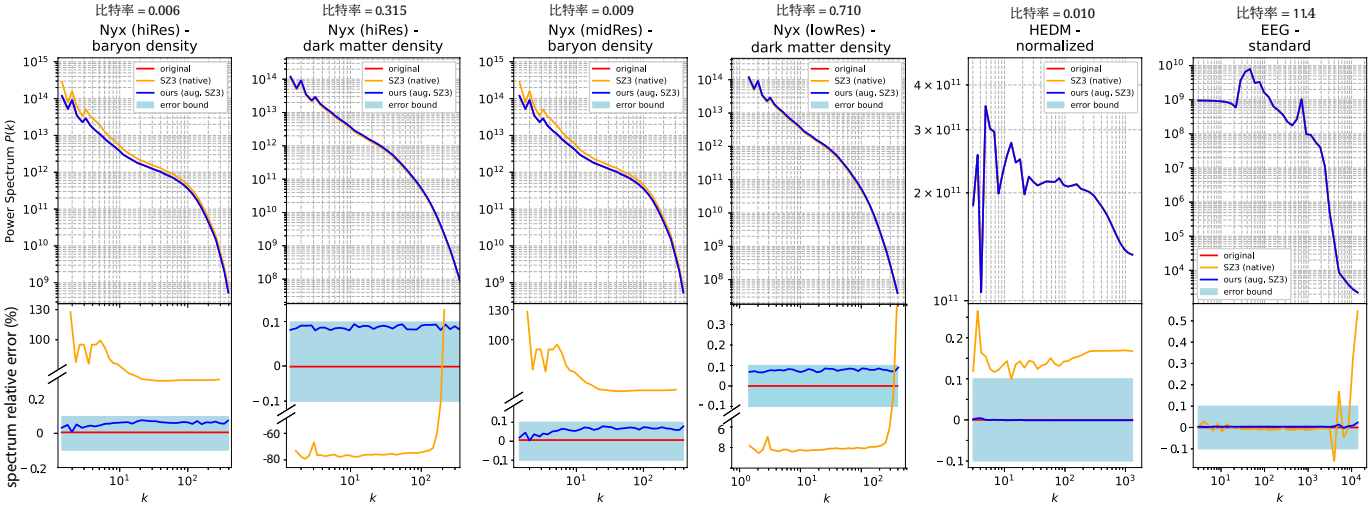


图10. SZ3 与采用我们方法增强后的 SZ3 的功率谱及其比值，其中比值为重建功率谱与原始功率谱逐元素相除的结果。对于我们的方法，功率谱分量的相对误差界设为 0.1%，如蓝色带状区域所示。

多个领域——空间、频谱、时间或多分辨率，拓宽了其在各类科学工作流中的适用范围。

致谢

本研究工作得到了美国能源部科学办公室先进科学计算研究（ASCR）的支持，合同编号为 DE-AC02-06CH11357 和 DE-SC0025677。本研究使用了国家能源研究科学计算中心（NERSC）的资源。该中心是美国能源部用户设施，使用的 NERSC 资助编号为 DDR-ERCAP 0034457。

参考文献

[1] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Luki´c, and E. Van Andel, “Nyx：一种用于计算宇宙学的大规模并行自适应网格细化代码”，《天体物理学杂志》，第765卷，第1期，第39页，2013年。[2] K. Zhao, S. Di, M. Dmitriev, T.-L. D. Tonellot, Z. Chen, and F. Cap- pello，“通过动态样条插值优化科学数据的误差有界有损压缩”，载于*IEEE*第*37*届国际数据工程会议（*ICDE*）论文集。IEEE，2021年，第1643–1654页。[3] L. Zhang, R. Li, C. Ren, S. Di, J. Liu, J. Huang, R. Underwood, P. Grosset, D. Tao, X. Liang 等，“LCP：通过粒子有损压缩增强科学数据管理”，《*ACM*数据管理会议论文集》，第3卷，第1期，第1–27页，2025年。[4] X. Zou, T. Lu, W. Xia, X. Wang, W. Zhang, H. Zhang, S. Di, D. Tao, and F. Cappello，“科学数据相对误差有界有损压缩的性能优化”，《*IEEE*并行与分布式系统汇刊》，第31卷，第7期，第1665–1680页，2020年。[5] Y.-H. Chen, N.-Y. Huang, Y.-H. Chu, M.-H. Li, R.-I. Chang, and C.-H. Wang，“无线传感器网络中的动态误差有界数据压缩与聚合”，载于*2012*年*IEEE*传感器会议。IEEE，2012年，第1–4页。[6] C. Ren, X. Liang, and H. Guo，“一种用于非结构化网格上误差有界科学数据压缩的预测遍历方法”，载于《计算机图形论坛》，第43卷，第3期。Wiley Online Library，2024年，第e15097页。[7] X. Wu, S. Di, C. Ren, P. Jiao, M. Xia, C. Wang, H. Guo, X. Liang, and F. Cappello，“实现非结构化科学数据的高效误差控制有损压缩”，载于*2025*年*IEEE*国际并行与分布式处理研讨会（*IPDPS*）。IEEE，2025年，第370–382页。

[8] X. Liang、S. Di、D. Tao、S. Li、B. Nicolae、Z. Chen 和 F. Cappello，“利用有损压缩提升科学模拟数据转储性能”，载于*2019*年*IEEE*国际集群计算会议（*CLUSTER*）。IEEE，2019年，第1–11页。[9] S. Di、J. Liu、K. Zhao、X. Liang、R. Underwood、Z. Zhang、M. Shah、Y. Huang、J. Huang、X. Yu等，“科学数据集误差有界有损压缩综述”，*ACM*计算调查，第57卷，第11期，第1–38页，2025年。[10] X. Liang、H. Guo、S. Di、F. Cappello、M. Raj、C. Liu、K. Ono、Z. Chen 和 T. Peterka，“面向特征保持的二维和三维向量场压缩”，载于*PacificVis*，2020年，第81–90页。[11] H. Mouri，“附着涡假说下壁面湍流中的两点相关”，流体力学杂志，第821卷，第343–357页，2017年。[12] M. Li 和 W. Chen，“用于脑电图分类的基于FFT的深度特征学习方法”，生物医学信号处理与控制，第66卷，第102492页，2021年。[13] X. Liang、K. Zhao、S. Di、S. Li、R. Underwood、A. M. Gok、J. Tian、J. Deng、J. C. Calhoun、D. Tao等，“SZ3：用于组合基于预测的误差有界有损压缩器的模块化框架”，*IEEE*大数据汇刊，第9卷，第2期，第485–498页，2022年。[14] P. Lindstrom 和 M. Isenburg，“浮点数据的快速高效压缩”，*IEEE* 可视化与计算机图形学汇刊，第12卷，第5期，第1245–1250页，2006年。[15] S. Li、P. Lindstrom 和 J. Clyne，“使用 SPERR 进行科学数据有损压缩 ”，载于 *2023* 年 *IEEE* 国际并行与分布式处理研讨会（*IPDPS*）论文集。IEEE，2023年，第1007–1017页。[16] P. Lindstrom，“定码率压缩浮点数组”，*IEEE* 可视化与计算机图形学汇刊，第20卷，第12期，第2674–2683页，2014年。[17] X. Liang、S. Di、D. Tao、S. Li、S. Li、H. Guo、Z. Chen 和 F. Cappello，“针对科学数据集高压缩比优化的误差控制有损压缩”，载于*2018*年*IEEE*国际大数据会议（*Big Data*）论文集，2018年，第438–447页。[18] D. Tao、S. Di、Z. Chen 和 F. Cappello，“基于多维预测和误差控制量化，显著提升科学数据集的有损压缩效果”，载于*2017*年*IEEE*国际并行与分布式处理研讨会（*IPDPS*）论文集。IEEE，2017年，第1129–1139页。[19] K. Zhao、S. Di、X. Liang、S. Li、D. Tao、Z. Chen 和 F. Cappello，“通过二阶预测和参数优化显著提升高性能计算数据集的有损压缩效果”，载于第*29*届国际高性能并行与分布式计算研讨会论文集，2020年，第89–100页。[20] J. Liu、P. Jiao、K. Zhao、X. Liang、S. Di 和 F. Cappello，“QPET：一种通用且可移植的关注量保持框架，用于

误差有界有损压缩”，arXiv预印本 arXiv:2412 02799，2024年。[21] Y. Li、X. Liang、B. Wang、Y. Qiu、L. Yan 和 H. Guo，“MSz：一种用于纠正误差有界有损压缩器中 Morse-Smale 分割的高效并行算法”，IEEE 可视化与计算机图形学汇刊，2024年。[22] N. Gorski、X. Liang、H. Guo、L. Yan 和 B. Wang，“为有损压缩器增添拓扑保证的通用框架”，IEEE 可视化与计算机图形学汇刊，2025年。[23] V. Bulgakov 和 A. Segal，“利用快速傅里叶变换降低句子转换器向量数据库的维度”，arXiv预印本 arXiv:2404.06278，2024年。[24] P. Buneman，“快速傅里叶变换作为数据库查询”，1993年。[25] D. Marten、H. Meyer 和 A. Heuer，“在 SQL 中计算傅里叶变换”，载于欧洲数据库与信息系统进展会议。Springer，2019年，第151–166页。[26] W. Chu 和 B. Champagne，“一种基于FFT、抗噪的听觉频谱及其在音频分类中的应用”，IEEE音频、语音与语言处理汇刊，第16卷，150，2007年1，第137页，第–期。[27] A. Celentano 和 V. Di Lecce，“基于FFT的图像签名生成技术”，载于图像与视频数据库存储和检索 V，第3022卷。SPIE，1997年，第457–466页。[28] W.-H. Tsai、Y.-M. Tu 和 C.-H. Ma，“一种用于哼唱检索系统的基于FFT的快速旋律比较方法”，模式识别快报，第33卷，第16期，第2285–2291页，2012年。[29] M. Asadzadehkaljahi、A. Halder、P. Shivakumara 和 U. Pal，“一种基于时空FFT的方法，用于受保护和敏感场景视频中的任意运动目标分类”，载于人工智能与应用，第3卷，第2期，2025年，第123–130页。[30] M. P. van Daalen、J. Schaye、C. Booth 和 C. Dalla Vecchia，“星系形成对物质功率谱的影响：精确宇宙学面临的挑战”，皇家天文学会月刊，第415卷，第4期，第3649–3665页，2011年。[31] A. E. Burgess 和 P. F. Judy，“幂律噪声中的信号检测：频谱指数的影响”，美国光学学会杂志 A，第24卷，第12期，第B52–B60页，2007年。[32] D. E. Sigeti，“噪声存在时确定性动力学的存续与高频功率谱的指数衰减”，物理评论 E，第52卷，第3期，第2443页，1995年。[33] S. Jin、J. Pulido、P. Grosset、J. Tian、D. Tao 和 J. Ahrens，“通过细粒度率质量建模，对宇宙学模拟中的原位有损压缩进行自适应配置”，载于第30届国际高性能并行与分布式计算研讨会论文集，2021年，第45–56页。[34] D. Sundararajan，《离散傅里叶变换：理论、算法与应用》。World Scientific，2001年。[35] NumPy 开发者，“numpy.fft.fftshift — NumPy v2.4.dev0 手册”，h
ttps://nu
mpy.org/devdocs/reference/generated/numpy.fft.fftshift.html，2008年，在线；访问日期：2025年7月8日。[36] A. Bienenstock 和 P. Ewald，“傅里叶空间的对称性”，Acta Crystallographica，第15卷，第12期，第1253–1261页，1962年。[37] D. A. Huffman，“一种构造最小冗余编码的方法”，IRE论文集，第40卷，第9期，第1098–1101页，1952年。[38] Y. Collet，“Zstandard（ZSTD）”，https://github.com/facebook/zstd，2015年，版本1.5.7，在线；访问日期：2025年6月15日。[39] NVIDIA 公司，“cuFFT：CUDA 快速傅里叶变换库”，https://docs.nvidia.com/cuda/cufft/index.html，2024年，版本12.4，在线；访问日期：2025年7月8日。[40] P. A. Penczek，“一类重建算法的三维谱信噪比”，结构生物学杂志，第138卷，第1–2期，第34–46页，2002年。[41] M. Unser、B. L. Trus 和 A. C. Steven，“一种基于谱信噪比的新分辨率判据”，Ultramicroscopy，第23卷，第1期，第39–51页，1987年。[42] M. Hassanzadeh 和 B. Shahrrava，“Parseval 定理的线性版本”，IEEE Access，第10卷，第27 230–27 241页，2022年。[43] F. Cappello、M. Ainsworth、J. Bessac、M. Burtscher、J. Y. Choi、E. Constantinescu、S. Di、H. Guo、P. Lindstrom 和 O. Tugluk，“SDRBench：科学数据约简基准”，https://sdrbench. github.io/，2020年，在线；访问日期：2025年7月8日。[44] A. Anand，“脑电图（EEG）数据集”，https://www.kaggle.com/datasets/amananandrai/complete-eeg-dataset，2020年，在线；访问日期：2025年7月8日。[45] 美国国家能源研究科学计算中心（NERSC），“Perlmutter 架构”，https://docs.nersc. gov/systems/perlmutter/architecture/，2015年，访问于2025年6月15日。[46] OpenMP 架构审查委员会，“OpenMP：OpenMP API 规范”，https://www.openmp.org/，2024年，访问于2025年7月8日。