Vorbis 和 Opus 音频文件的无损重新编码

Kamila Szewczyk

arXiv:2611.32641v1 [cs.SD] 24 Nov 2026

摘要——**我们介绍了 Balrogg，这是一种无损重压缩器，使用自适应上下文混合模型替代 Ogg Vorbis 和 Ogg Opus 文件中的熵编码。对于 Vorbis，我们将每个码本条目表示为由小型有符号整数（数字）组成的元组，在五个阶段中进行二值化，并与直接上下文模型、比特历史模型和匹配模型混合；模型权重由相邻数字和前一个残差遍共同决定。对于 Opus，每个符号都由自适应上下文模型进行编码，该模型以编解码器的概率表作为先验进行初始化；CELT 层的金字塔矢量量化器（PVQ）索引则被递归地拆分为脉冲计数。在默认级别下，Balrogg 可将 Vorbis 音乐压缩 16.2%、Vorbis 语音压缩 10.6%，并将 Opus 语音压缩 5.4%；xz、zstd、ZPAQ 和 bzip3 在相同文件上的压缩率最多提升 4.1%。在 48 种编码器配置下，文件大小可缩减至原大小的 66.5–96.1%。编码和解码相互对称，速度为每个核心 1.8–3.2 MB/s，并且使用的内存少于 12 MiB。**

索引词——**音频编码、上下文混合、无损重压缩、Opus、金字塔矢量量化、区间编码、重压缩、Vorbis。**

I. 引言

VORBIS [1] 和 Opus [2] 使用简单、快速的熵编码器。Vorbis 使用静态霍夫曼编码对矢量量化器码本条目进行编码。Opus 使用带有固定概率表的区间编码器，并以均匀分布对其 CELT 层 [3], [4] 的金字塔矢量量化（PVQ）索引进行编码。这两种编解码器都不会根据录音的统计数据调整其熵编码。

无损重压缩器会还原熵编码，并使用更强的模型再次对符号进行编码。解码器会恢复原始文件，因此校验和与签名仍然有效；这适用于归档和传输。

packMP3 [7] 是 MP3 重压缩方面的先例。packJPG 和 Lepton [5], [6] 可将 JPEG 文件无损压缩约 20–23%。OptiVorbis [8] 会重新计算 Vorbis 文件的霍夫曼编码，并写入一个更小且可播放的文件；该文件不同于输入文件（因此，从某种意义上说，它是无损重压缩的对偶）。对于第 V 节的按比特打包的 Vorbis 和 Opus 数据流表面上，通用压缩器只能节省约 0.1–4.1%的文件大小。

本文介绍了 Balrogg[1], 据我们所知，这是首个针对 Vorbis 和 Opus 的无损重压缩器。第 II 节概述了编码器。第 III 节介绍 Vorbis 编码模型：一种与码本无关的残差项表示，以及一种在线逻辑混合器，其权重由相邻数字和前一个残差遍选择。第 IV 节介绍 Opus 模型：带有预置先验且对其遗憾设有界限的自适应模型（命题 1）；对 PVQ 索引进行递归拆分，使编解码器对任意拆分树的码长保持不变（命题 2）。第 V 节报告了在五个数据集、48 种编码器配置以及来自互联网档案馆的随机样本上的大小、速度和内存占用。我们假设读者熟悉二元上下文混合 [9]–[11],区间编码[12],以及 Vorbis 和 Opus 比特流。

作者来自德国萨尔布吕肯的萨尔兰大学计算机科学系算法生物信息学组和数学系（电子邮箱：szewczyk@cs.uni-saarland.de）。[1] Balrogg 是基于 GNU GPLv3 发布的自由软件：https://github.com/iczelia/balrogg。所有陈述和测量均针对 1.5 版（提交 c7aedca）。

II. 编码器的结构

编码器将 Ogg 页面 [13], [14] 拆分为数据包并进行帧封装，然后解析每个数据包，直至其熵编码阶段。不执行任何变换。由此得到三类数据：

1）有效载荷符号，由第三节和第四节中的模型编码；2）容器数据（页面头部、分段值，以及标识头部和注释头部），由残差模型和字节模型编码；3）辅助信息：不同于其预测值的 Vorbis 类别词、非零填充、截断的数据包，以及不同于参考区间编码器输出的 Opus 帧字节（第四节 C 部分）。

二元决策使用带进位传播的 32 位区间编码器（采用 D. Subbotin 的方法）和 16 位概率。Opus 多符号决策使用 32 位区间编码器，其整数累积频率总和至多为 2^{17}。归档文件将编号的数据流以不超过 1 MiB 的分块进行复用，且不会重置编码器。所有模型仅使用整数运算。归档文件不存储模型参数，只有第 III-E 节的三个调节字节除外；其结果也不依赖于编译器、字长、字节序或向量指令集。

III. VORBIS

A. 残差数字

查找类型为 1 且维度为 d 的 Vorbis 码本具有整数乘数 $M[0]$、……、$M[n_v-1]$。它将条目

$$e=\sum_{j=0}^{d-1} a_j n_v^j, \qquad 0 \le a_j < n_v, \tag{1}$$

映射为向量（$M[a_0]$、……、$M[a_{d-1}]$），解码器会对其进行缩放并将其加到残差中。Balrogg 对其进行编码

$$v_j = M[a_j]-b, \qquad b=\left\lfloor \frac{1+\max_a M[a]}{2} \right\rfloor. \tag{2}$$

表 I 余数数字直接模型的上下文。c：位置，$\lambda(c)$：位长度，c，h：通道（截取为 3），m：内存，g：第一通道最新数字的幅度类别，$f=[g\neq 0]$，[时隙未改变]：先前编码的数字使用了相同的槽位。

阶段	上下文
0，$v\neq 0$	$\lfloor c/4\rfloor$，h，f，$[m=0]$，[时隙未改变]
1，符号	$\lambda(c)$，h、f、$[m\neq 0]$，m 的符号，h 中最后两个符号
2，$\lvert v\rvert=1$	$\lambda(c)$，h、g、$[m\neq 0]$，v 的符号
3，ℓ	h、g，三位树的节点
4，尾数	h、g、ℓ，最后两个尾数位

居中处理将所有码本的数字置于同一个整数网格上，无论其步长和码字长度如何，因此在一个码本上学习到的统计数据可以迁移到其他码本。解码器使用 M^{-1} 对（2）进行逆运算，通过（1）计算 e，并写入 e 的码字。

相似的码本共享模型。如果两个码本的条目数量、乘数数量和量化器步长均相同，并且其码字长度的平均差异不超过每个已使用条目 1 比特，则会为它们分配同一个模型槽位。槽位会跨链式链接持续存在，因此，重新定义已知码本的链接会重新使用其模型。

B. 二值化和直接模型

一个数字 v 会在最多五个阶段中编码为二元决策：(0) 是否 $v\neq 0$；(1) v的符号；(2) 是否 $|v|=1$；(3)$\ell=\lfloor\log_2|v|\rfloor-1\in\{0,\dots,7\}$，作为一棵三比特二叉树；(4) $|v|$ 的最高有效 1 以下的 $\ell+1$ 比特。如果 $v=0$，编码会在阶段 0 之后停止；如果 $|v|=1$，则会在阶段 2 之后停止。这些阶段涵盖$|v|<2^9$；libvorbis 写入 $|v|\leq 54$，FFmpeg 写入 $|v|\leq 8$。v 的幅度类别是 $|v|$的位长度：$v=0$为 0，$|v|=1$为 1，其他情况为 $\ell+2$。

每个决策都有一个直接上下文模型；也就是说，在由表 I 的上下文索引的表中，有一个自适应（指数移动平均）估计器。该估计器保存一个16位概率 P，表示比特为零的概率，以及一个计数 n。令 $r_n=\lfloor 65535/(n+3)\rfloor$。比特 z 编码后，

$$P\leftarrow\begin{cases}P+\lfloor(65535-P)\,r_n/2^{16}\rfloor, & z=0,\\ P-\lfloor P\,r_n/2^{16}\rfloor, & z=1,\end{cases}\tag{3}$$

并且 n递增，直到达到上限 $n_{\max}$（默认为 120）。因此，速率会逐渐衰减为 $1/(n+3)$，直至 $1/(n_{\max}+3)$。第 1 级仅使用直接模型。

设 c 为数字在其通道 h中的位置；对于类型2残差，需先进行去交错。除 c和 h外，上下文还使用两个更早的数字。

第一个是第一通道的最新数字，其幅度类别为 g，且 $f=[g\neq 0]$。由于立体声类型2残差会交错排列各通道的数据，因此对于 $h=1$，该数字是同一位置上另一个通道的数字；对于 $h=0$，则是前一位置上同一通道的数字。

第二个是前一个残差遍次中相同位置上的数字。对于每个残差、遍次、通道和位置，编码器都会保留一个字节，用来存储在那里编码的最后一个数字的符号和幅度类别。

在此处编码的最后一个数字。数字在遍次 $\tau>0$中的内存 m，是其位置上遍次 $\tau-1$ 的记录；在第一遍中，它是较早数据包的第一遍留下的记录。

C. 混合

从第 2 级开始，在线逻辑混合器 [9]–[11]根据六个输入 t_1、.. .、t_6计算决策的概率。令 $\mathrm{st}(p)=\ln\frac{p}{1-p}$，并令 $\mathrm{sq}=\mathrm{st}^{-1}$。使用由上下文按下文所述选出的权重向量 w，比特为 1 的概率为 $p=\mathrm{sq}(\sum_i w_i t_i)$。比特 y 编码后，

$$w_i\leftarrow w_i+\eta\,2^{-11}(y-p)\,t_i,\tag{4}$$

学习率 $\eta=7$ 默认为该值。输入如下：

1) $t_1=\mathrm{st}(1-P)$，直接模型的预测；2) $t_2=\mathrm{st}(p_s)$，比特历史模型的预测。每个阶段都有一张包含 2^{18} 单字节状态的表，其索引由槽位 h、c 中的符号以及 m 中幅度类别的低两位计算出的哈希确定。自适应状态映射给出状态 s下比特为 1 的概率 p_s；3) $t_3=1$，偏置；4) $t_4=\varepsilon t_2$，当历史 s 只包含 0 时取 $\varepsilon=+1$，只包含 1 时取 -1，否则取 0；5) $t_5=\pm\mathrm{st}(\alpha)$，其中 α 是匹配模型正确预测该比特的概率，该概率按阶段和匹配长度学习；6) $t_6=\pm\min(L、32)/4$，其中 L 是匹配长度。

这些状态是 [9]，的 253 个状态，它们能够精确地统计少量的n_0、n_1个零和一，并近似地统计较大的数量。状态映射从 $(n_1+1)/(n_0+n_1+2)$ 开始，对于单个比特值的历史记录采用更有把握的值，并通过 (3) 进行更新。

该混合器与通用上下文混合压缩器中的混合器有三点不同。

- 权重选择。每个阶段有 200 个权重向量。一个向量由四个值选定：(i) 前一个相同通道数字的幅度类别（截断为 4）；如果该数字不在位置 $c-1$，则为 0；(ii) 最近编码的数字的截断幅度类别（如果它属于其他通道），否则为 0；(iii) h 的奇偶性；(iv) m 中幅度类别的低两位。

- 初始化。每个权重向量都初始化为 (1、0、.....、0)，因此混合器最初会复现直接模型，即第 1 级代码；只有当其他输入的表现优于直接模型时，它们才会获得权重。这可以防止短文件在“维数灾难”中出现欠拟合。

- 数字级匹配模型。匹配模型预测完整数字，而非字节或比特。它存储最近的 2_{18} 个数字（截断至 $[-127,127]$），以及一张包含 2_{16} 个位置的表，这些位置按最近十个数字的滚动哈希进行索引。通过比较候选项之前的数字来验证候选项。如果至少十个数字一致，则候选项后面的数字就是预测值，匹配长度 L 计数最高可达 63。每个阶段都会接收由预测数字所隐含的比特，直到首次不匹配为止。匹配模型默认关闭

第 III-E 节中的搜索可能会为某个文件启用该模型。

混合器使用 16 位定点算术。对数几率以 $1/256$ 为单位存储，并限制在 $|x| < 8$；权重以 2^{-15} 为单位存储，并会饱和；误差 $y - p$ 占 12 位。函数 sq 在 33 个点上进行表格化，并采用线性插值。st 是 (sq) 的查表逆函数。

D. floor、类别和异常数据包

类型 1 floor 的值按频率升序编码。每个值都编码为一个零标志、一个位长度和若干尾数位，并将 floor 编号和 post 作为上下文。此外，零标志使用前两个标志；位长度（三位树）使用同一 post 处的前一个位长度；尾数位则使用其位置和前两个比特。

多个类别词可以编码同一个 floor 分区。编码器会预测那个针对每个值选择首个能够表示该值的子类码表的类别词，并且仅当类别词与预测不同时才对其编码。

残差分区的类别编码为一棵四比特树，其上下文包括分区索引和残差编号。如果第 III-E 节的搜索启用该功能，上下文还包括前一个数据包中同一分区的类别，并将其缩减为六个桶。

每个数据包进行预处理扫描，以检测四种异常情况：误预测的类别字、非零填充、尾随位和截断。数据包类型符号会标记这些情况。尾随位（包括最后一个不完整的字节）会单独编码，而截断的数据包则使用自适应字节模型进行编码。

E. 级别和参数搜索

第1级至第4级的区别在于所混合的阶段：第1级不混合任何阶段，第2级混合阶段 0 和 2，第3级混合阶段 0 和 2–4，第4级则混合全部五个阶段。其他阶段使用各自的直接模型。第5级至第9级使用第4级的模型，并通过额外编码整个文件 2、4、6、8 和 12 遍来搜索更优参数。搜索采用贪心策略。它首先尝试默认关闭的两个开关：前一数据包的类别上下文和匹配模型。然后按此顺序尝试 $n_{\max} \in \{40, 200, 61, 255, 90, 25, 150\}$ 和 $\eta \in \{10, 5, 3\}$。每次尝试都会更改当前最佳设置中的一个参数，只有在归档文件变小的情况下才会保留该更改。选定的设置占用三个字节，第4级至第9级的解码速度相同。

IV. Opus

A. 以编解码器为先验的自适应模型

Balrogg 使用精简版的 libopus 解码器解析 Opus 数据包，该解码器保留所有会影响熵编码的分支，并省略合成步骤。每次调用范围解码器时，都会报告解码值、符号类型、该符号在语法中的位置（位置），以及它在编解码器中的分布。Balrogg 使用以该分布初始化的自适应模型重新编码该值。令 p_i 为

表 II Opus 多符号模型：先验、尺度 S、增量 δ、重新缩放阈值 H和上下文。

类型（先验）	S	δ	H	上下文
表格化（表）	384	28	2^{14}	表、位置、位置处的前一个值（截断为 7）
粗能量（拉普拉斯）	512	48	2^{14}	频带、通道、帧内标志、帧大小；前一帧的残差，前一频带、其他通道（5 每个类别）
细能量（均匀）	1024	16	2^{14}	频带、通道，　宽度；前一个值；其他通道（5 个类别）
波段角度（步进、三角化）	192	32	2^{14}	频带、概率密度函数、波段大小类别、分辨率；前一帧角度（8 个类别）
小整数（均匀）	1024	24	2^{13}	位置、字母表大小
脉冲分割，公式（8）	1024	16	2^{15}	频带、深度（N_1，　N_2，　K）

编解码器中符号 i 的概率，令 S 为一个尺度。新上下文以先验频率

$$f_i = \max\{1, \min\{60000, \lfloor S p_i + 1/2 \rfloor\}\} \tag{5}$$

并以概率 i对符号进行编码

$$q(i) = \frac{f_i + n_i}{F + \sum_j n_j}, \tag{6}$$

其中，$F = \sum_i f_i$ ，且计数 n_i 从零开始。对符号 i 进行编码会向 n_i增加增量 δ 。当 $\sum_j n_j$ 超过阈值 H时，每个 n_j 都会被重新缩放。

公式（6）是 Krichevsky–Trofimov 估计器 [15]，只是用编解码器的表替代了均匀伪计数。

表 II 列出了每种符号的参数和上下文。编解码器表通过其内容的哈希进行标识，因此相同的表会共享统计数据。SILK 帧只包含表格化符号，因此表 II 的第一行涵盖整个 SILK 层。在编解码器中，概率为 $2^{-\kappa}$ 的二元决策使用比特模型（3），并将该概率作为初始值，同时以位置、 κ以及该位置上前两个结果作为上下文。

粗能量残差在 $[-24, 24]$ 上建模；其他值则编码为转义符号、符号和 16 比特幅值。相邻残差以五个类别进入上下文： ≤ -2、 -1、0、1 和 ≥ 2。Opus 将频带的细能量比特作为原始比特写入；Balrogg 则将其编码为具有均匀先验的单个符号，使模型能够利用跨帧和跨声道的细能量相关性。范围超过 1024 的均匀整数以及其他原始比特（单系数频带的符号、后滤波参数和防坍缩标志）均使用均匀分布进行编码。

从编解码器的表开始所产生的代价是有界的。每次观测都会增加相同的增量，因此（6）是一个波利亚 urn。考虑一个上下文在其第一次重新缩放之前的情况。令 $\alpha_i = f_i/\delta$和 $A = \sum_i \alpha_i$。设一个包含 u 个符号的序列中，符号 i 恰好出现 u_i 次，并令 $x^{(n)} = x(x+1)\cdots(x+n-1)$。根据（6），该序列的理想码长为

$$L = \log_2 \frac{A^{(u)}}{\prod_i \alpha_i^{(u_i)}}, \tag{7}$$

这与符号的顺序无关。使用编解码器的量化表 $\tilde{p}_i = f_i/F$ 的静态码，其码长为 $L_0 = \sum_i u_i \log_2(A/\alpha_i)$。

命题 *1*：对于在首次重新缩放之前于同一上下文中编码的每个 u 符号序列，

$$L - L_0 \le \frac{1}{2}\sum_{i:u_i>0} \log_2\Big(1+\frac{u_i}{\alpha_i}\Big) - \frac{1}{2}\log_2\Big(1+\frac{u}{A}\Big) + \log_2 e\Big(\sum_{i:u_i>0}\frac{1}{12\alpha_i} + \frac{1}{12(A+u)}\Big).$$

证明：根据 (7)，$(L - L_0)\ln 2 = h(A,u) - \sum_i h(\alpha_i, u_i)$，其中 $h(x,n) = \ln x^{(n)} - n\ln x$。由斯特林公式 $\ln\Gamma(x) = (x-\frac{1}{2})\ln x - x + \frac{1}{2}\ln 2\pi + \mu(x)$，$0 < \mu(x) < 1/(12x)$，可得 $h(x,n) = x\psi(n/x) - \frac{1}{2}\ln(1+n/x) + \mu(x+n) - \mu(x)$，其中凸函数 $\psi(r) = (1+r)\ln(1+r) - r$。根据詹森不等式，$\sum_i \alpha_i \psi(u_i/\alpha_i) \ge A\psi(u/A)$。含有 $u_i = 0$ 的项为零，而关于 μ 的界给出了所要证明的结论。 ∎

令 r 表示序列中出现的符号值数量。由于 $\alpha_i \ge 1/\delta$，右侧最多为 $\frac{r}{2}\log_2(1+u\delta/r) + \frac{r\delta+1}{12}\log_2 e$。因此，与量化表相比，上下文学习每个出现的值最多额外花费约 $\frac{1}{2}\log_2(u\delta/r)$ 比特；而拟合数据更好的模型，其收益会随 u 线性增长。这个界几乎是紧的。对于有九种可能计数且先验频率相等的脉冲分割，该界表示在上下文的前 $H/\delta = 2048$ 个符号上多花费 20.3 比特，即每个符号 0.01 比特；最坏情况下，序列比静态码多花费 20.3 比特。证明只使用 $\alpha_i > 0$，因此重新缩放后，相对于该时点的模型分布，这个界对下一个周期仍然成立。

B. PVQ 索引的递归编码

CELT 将频带的形状编码为 $\mathcal{V}(N, K) = \{y \in \mathbb{Z}^N$ 中某个向量的索引：$\sum_j |y_j| = K\}$，即含有 K 个脉冲的整数向量，并为每个 $V(N, K) = |\mathcal{V}(N, K)|$向量赋予相同的概率。$V(N, K)$呈组合式增长，因此模型无法为每个索引分别保存计数。Balrogg 则将索引解码为对应的向量，并将坐标划分为 $N_1 = \lfloor N/2 \rfloor$ 和 $N_2 = N - N_1$ 两组。它使用先验为下式的 (6) 型模型，对第一组中的脉冲数 k 进行编码：

$$p(k \mid N_1, N_2, K) = \frac{V(N_1,k)\,V(N_2,K-k)}{V(N,K)}, \quad 0 \le k \le K. \tag{8}$$

这两组分别含有 k 和 $K-k$ 个脉冲，并以相同的方式继续划分，直至达到深度上限。超过该上限后，每个子向量都以其索引为依据，按照均匀分布进行编码。$V(N, K)$ 的表涵盖 $N, K \le 208$。如果频带的字母表大小与表中的值不同，则直接按照均匀分布对索引进行编码。

命题 *2*：令 $\mathcal{V}(N, K)$ 的坐标在任意位置递归地进行分割，直至任意叶节点。如果每次分割都使用概率 (8) 进行编码，并且每个叶节点都使用其均匀索引进行编码，那么每个 $y \in \mathcal{V}(N, K)$ 的概率为 $1/V(N, K)$。

证明通过对树进行归纳法，从恒等式 $\sum_k V(N_1, k)\,V(N_2, K-k) = V(N, K)$ 开始。例如，$y = (1, 0, -1, 0, 0, 2, 0, 0) \in \mathcal{V}(8, 4)$ 在深度 2 处具有分割概率 $\frac{32\cdot 32}{2816}, \frac{4\cdot 4}{32}, \frac{8\cdot 1}{32}, \frac{1}{4}$、$\frac{1}{4}, \frac{1}{8}$、$1$ 和，以及叶节点概率和。它们的乘积为 $1/2816 = 1/V(8,4)$。

分割模型从（8）开始，经（5）舍入，因此所有节省都来自自适应。实际的频带形状并非均匀分布，模型会针对每个频带、深度和（N_1, N_2, K）分别学习其分布（表 II 的最后一行）。级别设定了深度上限：级别 1 至 7 使用深度 0 至 6，而级别 8 和 9 使用深度 6。

树之前有一个重复标志。对于每个频带以及 N 和 K 的粗粒度类别，编码器会存储上一个索引。如果 $V(N, K) > 16$ 且字母表大小与所存储索引的相同，则由一个自适应比特指示当前索引是否等于所存储的索引。如果相等，则不再编码其他内容。该树不依赖于该标志，因此一个频带的所有向量共享分割统计信息。

C. 帧字节的精确恢复

再次编码解码符号并不总能还原原始帧。区间编码器可以自由选择其末尾的一些比特。区间编码数据从帧的开头写入，而原始比特从末尾写入。两者之间的字节内容是任意的，FFmpeg 会将较早帧的字节留在那里。因此，Balrogg 的编码器和解码器使用 libopus 范围编码器的副本，即镜像编码器，将符号编码到一个长度等于该帧长度的缓冲区中。设 x 为原始帧，$\hat{x}$ 为镜像编码器的输出。自适应标志指示 $x \ne \hat{x}$ 是否成立。若成立，编码器存储 $a = \min\{j : x_j \ne \hat{x}_j\}$、$b = \max\{j : x_j \ne \hat{x}_j\}$ 以及字节 $x_a, \ldots, x_b$，解码器则使用这些字节覆写 $\hat{x}_a, \ldots, \hat{x}_b$。该区间可以覆盖整个帧，因此任何能够解析的帧都可以被表示。

位置编码为 a 和 $b-a$。每个存储的字节首先根据前四帧中相同偏移处的值进行预测，仅当这些预测均不正确时，才将其编码为字面值。供应商字符串会选择三个编码器类别（libopus、FFmpeg、其他）之一，作为这些模型的上下文。该标志的初始概率为 2^{-12}。对于 FFmpeg 数据流，该标志从 $1/2$开始，而 a 和 $b-a$ 则编码为相对于前一帧的差值。镜像编码器重现了第 V 节 MLS 文件中的全部 2 090 820 帧，以及第 V-C 节 760 万个 libopus 帧中除 104 帧以外的全部帧。目录字节、帧数、帧长度和数据包填充各自拥有专用模型。

V. 测量

A. 设置

已验证所有归档文件均可解码还原为其输入内容。Balrogg 使用 GCC 15.2 在 -O2 下编译。基线包括 xz 5.8.3（-9e） [16], zstd 1.5.7（-19） [17],ZPAQ 7.15（-m5，其最高级别） [18],以及 bzip3 1.5.4（每个文件一个块）[19],，并分别应用于每个文件。测试机器配备 16 核、32 线程的 AMD Ryzen 9 5950X 处理器和 64 GB 内存，运行 Linux 7.0。文件从页缓存中读取，并写入内存支持的文件系统，而且

表 III 完整数据集在级别 1、4 和 9 下的压缩大小（%），以及对每个文件分别应用 xz 和 zstd 后的压缩大小（%）。时间为使用 32 个进程在第4级处理整个集合所需的墙钟时间（秒）。内存为任意级别下每进程峰值中最大的值，单位为 MiB。

				Balrogg（%）			基线（%）		第4级		
Set	编解码器	文件	MB	L1	L4	L9	xz -9e	zstd -19	编码 (秒)	解码 (秒)	Mem.
OpenMIC-2018	Vorbis	20 000	2595.8	86.2	83.8	83.5	97.5	97.5	68.4	71.5	6.6
VoxPopuli	Vorbis	41	5178.9	89.8	89.4	89.3	98.6	98.3	120.6	129.7	11.7
Minecraft (V)	Vorbis	741	29.5	83.9	80.8	80.4	96.9	96.9	2.4	2.5	8.6
钢琴曲 (P)	Vorbis	1	2.7	83.3	79.0	78.4	97.4	97.6	1.1	1.2	5.5
MLS	Opus	2729	182.1	94.9	94.6	94.3	99.5	98.6	3.4	3.5	4.7

表 IV 相同文件的单进程比较（OpenMIC-2018：前 1000 个文件；VoxPopuli：前两段录音）：压缩大小（%）和吞吐量（MB/s），编码/解码。

	压缩大小（%）						吞吐量，编码/解码（MB/s）					
Set	L1	L4	xz -9e	zstd -19	ZPAQ	bzip3	L1	L4	xz -9e	zstd -19	ZPAQ	bzip3
OpenMIC-2018	86.7	84.5	97.8	97.7	97.4	97.4	4.9/5.0	2.3/2.2	1.4/15.5	6.5/22.9	0.22/0.22	5.5/5.2
VoxPopuli	91.2	90.8	99.9	99.7	98.6	98.6	7.1/7.1	3.2/2.9	2.4/331	11.7/3320	0.33/0.33	5.9/4.9
集合 V	83.9	80.8	96.9	96.9	97.8	96.6	3.3/3.4	1.9/1.8	1.0/6.4	4.1/7.5	0.16/0.16	3.6/3.5
集合 P	83.3	79.0	97.4	97.6	95.9	96.2	6.0/6.3	2.5/2.4	4.6/35.8	10.7/189	0.23/0.24	6.6/6.0
MLS	94.9	94.6	99.5	98.6	99.6	98.8	3.7/3.9	2.7/3.0	0.9/9.8	5.5/12.4	0.19/0.19	4.5/4.4

表 V 表 IV 中的文件的编码/解码每进程内存（MiB）：集合中文件的最大峰值。

Set	L1	L4	xz -9e	zstd -19	ZPAQ	bzip3
OpenMIC	2.0/1.9	3.5/3.4	66/0.2	5.6/1.1	85/84	5.9/5.9
VoxPopuli	6.5/3.6	8.0/5.1	673/64	713/11	976/849	1278/1277
集合 V	2.9/2.8	4.4/4.3	86/2.4	56/4.9	127/126	18/18
集合 P	2.4/2.4	4.0/3.9	87/2.7	56/4.7	128/126	19/18
MLS	4.3/4.3	4.7/4.7	65/0.2	2.6/0.5	84/84	5.6/5.5

表 VI 第4级各文件的压缩大小（%）：中位数、四分位距、第 5 至第 95 百分位数以及范围。

Set	中位数	IQR	第 5–95 百分位数	范围
OpenMIC-2018	84.8	81.6–86.8	74.7–88.5	35.5–92.5
VoxPopuli	89.9	88.7–90.7	86.9–90.9	86.0–91.0
Minecraft (五)	78.9	74.0–83.3	65.8–88.1	52.4–92.0
MLS	94.6	94.2–94.8	93.8–95.1	92.5–95.3

每个文件均由单个单线程进程进行编码。内存是进程匿名常驻内存的峰值。

我们使用了五个数据集。OpenMIC-2018 [20] 包含 20000 段时长为十秒、采用 Vorbis 编码、码率约为 112 kb/s 的立体声音乐片段。VoxPopuli [21] 中未标注的 2020 年英语部分包含 41 段单声道 Vorbis 录音，文件大小最大为 220 MB，码率约为 48 kb/s。多语言 LibriSpeech（MLS） [22] 的波兰语和葡萄牙语测试集与开发集包含 2729 段 Opus 语音，平均时长为 15 秒，码率约为 35 kb/s。集合 V 包含 *Minecraft* 1.2. 中的 741 个 Vorbis 音效和音乐文件，其中 449 个各不相同；集合 P 则是同一款游戏中的一条钢琴音轨。我们在集合 V 和集合 P 上开发模型，只有在参数固定后才处理所有其他数据。

B. 压缩、速度和内存

在第4级（表 III），Balrogg 将 Vorbis 音乐压缩了 16.2–21.0%，将 Vorbis 语音压缩了 10.6%，并将 MLS 的 Opus 语音压缩了 5.4%。对于相同的文件，xz 和 zstd 可再压缩 0.5–3.1%。没有文件变得更大。第 9 级会对每个 Vorbis 文件编码 13 次，相较于第 4 级，压缩率提高 0.1–0.6 个百分点。在第4级使用 32 个进程时，该机器对 OpenMIC-2018 的编码速度为 38 MB/s，对 VoxPopuli 的编码速度为 43 MB/s。

对于使用单个进程处理的相同文件（表 IV 和 V），ZPAQ 在最高级别下可减少 0.4–4.1%，bzip3 可减少 1.2–3.8%。Balrogg 在第 4 级下可减少 5.4–21.0%，速度约为 ZPAQ 的十倍

表 VII：OpenMIC-2018 的 2000 个文件（254.5 MB），第4级：挂钟时间、加速比，以及所有运行中的进程私有内存总和的最大值。三次运行的均值（变异系数至多为 0.25%）。

Jobs	编码（秒）	解码（秒）	加速比		内存（MiB）
			Enc.	Dec.	
1	109.7	110.0	1.0	1.0	3.5
2	55.2	55.8	2.0	2.0	6.8
4	27.6	27.9	4.0	3.9	13.4
8	13.9	14.0	7.9	7.9	26.5
16	7.7	7.4	14.3	15.0	50.9
32	6.3	5.9	17.4	18.8	91.8

，且编码时仅使用 3.5–8.0 MiB 内存，即使处理 220 MB 的录音也是如此。

表 VI 给出了逐文件分布。文件是独立编码的，因此加速比几乎随物理核心数线性增长，直至 16 个物理核心（表 VII）。

C. 编码器、比特率和编码模式

我们使用 libvorbis 1.3.7、libopus 1.5.2 和 1.3.1，以及 FFmpeg 7.1 的 Vorbis 和 Opus 编码器，在 48 种配置下对 EBU SQAM 集的 70 条音轨进行了编码 [24] （表 VIII）。Balrogg 在级别 1、4 和 9 下均接受了全部 3360 个文件。在第 4 级下，Vorbis 的压缩大小为原来的 66.5–90.5%，Opus 则为 76.6–96.1%

表 VIII EBU SQAM，每种配置 70 个音轨：级别 1、4 和 9 的压缩大小（%）、第4级的单文件中位数，以及每个文件的 xz -9e；对于 Opus，给出第4级熵编码载荷的缩减比例（%）。LV：LIBVORBIS 1.3.7（质量 q）；FF：FFMPEG 7.1（Vorbis 的质量 q，Opus 的 KB/S）；LO：LIBOPUS 1.5.2，LO′：LIBOPUS 1.3.1（KB/S；立体声、20 毫秒帧，以及除非另有说明均采用自动模式）。

配置	L1	L4	L9 中位数	xz
LV q=0	82.1	77.5	76.3	77.8
LV q=2	85.5	82.2	81.4	84.1
LV q=4	84.7	82.4	81.8	83.6
LV q=6	87.8	86.2	85.9	87.4
LV q=8	89.1	87.7	87.4	88.8
LV q=10	91.5	90.5	90.3	91.8
LV q=2，单目	85.9	84.0	83.4	86.0
LV q=6，单目	88.5	86.7	86.3	88.7
LV q=4，5.1	80.1	76.9	73.7	77.6
LV CBR 128	73.7	71.4	70.9	71.7
FF q=2	69.0	66.5	65.0	65.8
FF q=5	77.0	74.8	73.7	74.5
FF q=8	79.3	77.3	76.3	77.4

配置	L1	L4	L9 Med.	xz	Payl.
LO 16	86.1	83.5	83.3	85.9	10.6
LO 24	87.0	85.4	85.4	87.8	11.4
LO 32	89.5	88.1	88.0	90.0	9.4
LO 48	92.1	91.0	90.9	92.2	7.2
LO 64	93.4	92.5	92.2	93.5	6.0
LO 96	94.5	93.8	93.5	94.5	5.0
LO 128	95.1	94.4	94.1	95.2	4.3
LO 192	96.1	95.4	95.1	96.2	3.6
LO 256	96.8	96.1	96.0	97.1	2.8
LO SILK 12	78.2	78.2	78.2	78.0	13.5
LO SILK 16	82.6	82.6	82.6	82.4	10.6
LO SILK 24	83.3	83.3	83.3	83.2	12.1
LO SILK 32	86.3	86.3	86.3	86.2	10.0
LO 混合 24	76.7	76.6	76.6	76.4	19.9
LO 混合 32	80.6	80.5	80.5	80.0	16.7
LO hybrid48	85.9	85.6	85.4	85.3	12.4
LO CELT 16	86.1	83.5	83.3	85.9	10.6
LO CELT32	89.5	88.1	88.0	90.0	9.4

配置	L1	L4	L9 Med.	xz	Payl.
LO CELT 64	93.4	92.5	92.2	93.5	6.0
LO CELT128	95.1	94.4	94.1	95.2	4.3
LO 24，单目	89.0	87.3	87.1	89.5	9.3
LO 64，单目	94.5	93.6	93.3	94.7	4.7
LO 64、2.5 毫秒	77.0	76.8	76.8	78.3	15.4
LO 64，5 毫秒	84.2	83.5	83.5	86.3	12.5
LO 64，10 毫秒	90.8	89.7	89.6	91.2	7.9
LO 64，40 毫秒	92.6	91.5	91.3	93.0	7.0
LO 64，60 毫秒	92.6	91.5	91.3	93.0	7.0
LO 64，CBR	80.7	79.7	79.5	76.9	6.6
LO 24，语音	78.7	78.6	78.6	78.0	17.6
LO′ 32	89.5	88.1	88.0	89.9	9.4
LO′ 64	93.4	92.5	92.2	93.5	6.0
LO′ 128	95.1	94.4	94.1	95.2	4.3
FF 64	81.0	79.9	79.6	78.5	18.1
FF 128	82.4	81.4	81.1	79.2	12.7
FF 256	83.6	82.8	82.7	80.3	7.6

表 IX 利用文件间冗余的基线：整个集合的压缩大小（%）。SETV 仅限于 ITS449 不同文件。

方法	V	OpenMIC	MLS
xz -9e，逐文件	97.2	97.5	99.5
zstd -19，逐文件	97.2	97.5	98.6
xz -9e ，实心归档	92.5	95.5	99.0
zstd -19 -long=31，实心归档	92.5	95.6	98.1
zstd -19，训练字典	92.7	95.5	98.3
Balrogg 第 4 级，逐文件	80.7	83.8	94.6
Balrogg 第 4 级，链式流	77.4	82.8	93.6

表 X 压缩大小（%）：OptiVorbis、第4级 Balrogg，以及将第4级 Balrogg 应用于 OptiVorbis 的输出。

Set	OptiVorbis	Balrogg	Both
OpenMIC-2018	94.5	83.8	83.5
VoxPopuli	97.1	89.4	89.4
Minecraft（V）	92.7	80.8	78.7
钢琴曲（P）	96.4	79.0	79.0

表 XI 互联网档案馆第4级文件：已检查并接受的文件、接受文件的压缩大小（%）及文件的95%自助法区间和单文件中位数。

集合	文件	接受	大小	95%区间	中位数
Opus，随机样本					
所有文件	1002	1000	94.1	94.0–94.2	94.3
FFmpeg	837	837	94.1	93.9–94.2	94.3
libopus	107	107	94.6	94.0–95.1	95.2
WhatsApp	35	34	91.9	91.5–92.4	92.5
其他编码器	23	22	91.3	88.4–93.8	88.5
Vorbis、随机样本、已上传文件					
所有文件	493	468	85.0	84.3–85.7	87.1
libvorbis 2001–2007	126	119	85.2	83.7–86.5	85.9
libvorbis 2009–2020	264	246	84.9	84.0–85.8	87.1
FFmpeg	79	79	84.2	82.7–85.7	88.6
aoTuV	22	22	88.0	84.3–90.8	90.1
Vorbis，随机样本，源自存档的文件					
Allfiles	1011	880	83.9	83.6–84.3	84.4
Opus，较早的集合（不同文件）					
所有文件	3727	3725	95.2	95.1–95.2	95.6
一个播客	2848	2848	95.4	95.4–95.4	95.7
其他项目	879	877	93.8	93.4–94.2	94.6

。Opus 的压缩率最高。压缩率提升幅度在 FFmpeg 和低质量 Vorbis 编码中最大。对于 Opus，该表还给出了熵编码载荷的缩减比例，该载荷不包括帧封装、头部和填充。使用 libopus 时，SILK 和混合模式下的载荷缩小了 10–20%，CELT 模式下则缩小了 2.8–15%。

D. 使用文件间冗余的基线

表 IX 增加了利用文件间冗余的基线，例如 libvorbis 在每个文件中重复使用的设置头部：整个集合的固态 tar 归档，以及使用在该集合上训练的 1-MiB 字典的 zstd，其大小也计入其中。分别对每个文件进行编码时，在两个 Vorbis 集合上，Balrogg 仍比这些基线中最优者小 11.8 和 11.7 个百分点，在 MLS 上则小 3.5 个百分点。Balrogg 还可以将文件串联编码为一个链式流，只存储一次重复的设置头部，并在各个链接之间共享模型（表的最后一行）。

与 OptiVorbis 0.3.2（表 X）相比，Balrogg 节省的空间是其 2.6 至 5.8 倍，而且还能压缩 OptiVorbis 的输出。

E. 来自互联网档案馆的文件

我们使用固定种子从互联网档案馆中抽取了随机样本。Opus 样本是最早上传的 .opus 每个音频项目的文件，共 1002 个。Vorbis 样本来自知识共享项目，其中包含 Vorbisfile：493 个已上传文件，以及归档文件使用自有转换服务生成的 1011 个文件。另行列出较早的一个包含 3727 个不同 Opus 文件的集合，因为其中 2848 个文件属于同一个

播客。没有文件变大（表 XI）。Opus 样本的压缩大小为 94.1%，Vorbis 样本为 83.9–85.0%；这些样本包含由 2001 年至 2020 年间 17 个版本的 libvorbis 写入的文件。所有未被接受的文件都存在容器损坏问题（最后一页不完整，或有字节位于任何页之外）。

VI. 结论

我们介绍了 Balrogg，这是一种适用于 Vorbis 和 Opus 的无损重压缩器。对于 Vorbis，它使用逻辑混合器对居中的乘数位进行编码；混合器从直接模型开始，并根据相邻数字和前一个残差遍的情况选择权重。对于 Opus，它使用自适应模型对每个符号进行编码，该模型以编解码器的表为先验，每个符号值的学习成本约为$1\ 2\ \log_2 u$比特；同时递归拆分 PVQ 索引，且不改变编解码器的码长。在已发表的数据集上，Balrogg 可将 Vorbis 文件压缩 10.6–21.0%，将 Opus 语音压缩 5.4%；在 48 种编码器配置下，压缩率为 3.9–33.5%；在从互联网档案馆抽取的随机样本上，Opus 和 Vorbis 的压缩率分别为 5.9% 和 15.0–16.1%。它在两个方向上的运行速度为每个核心 1.8–3.2 MB/s，使用的私有内存少于 12 MiB。

参考文献

[1] Xiph.Org 基金会，“Vorbis I 规范”，2020年7月。 [在线]。获取地址：https://xiph.org/vorbis/doc/Vorbis_I_spec.html [2] J.-M. Valin、K. Vos 和 T. B. Terriberry，“Opus 音频编解码器的定义”，RFC 6716，2012年9月，doi：10.17487/RFC6716。[3] T. R. Fischer，“一种金字塔矢量量化器”，*IEEE 信息论汇刊*，第32卷，第4期，第568–583页，1986年7月，doi：10.1109/TIT.1986.1057198。[4] J.-M. Valin、G. Maxwell、T. B. Terriberry 和 K. Vos，“Opus 编解码器中的高质量低延迟音乐编码”，载于第*135*届 *AES* 大会论文集，美国纽约州纽约市，2013年10月，论文8942。[5] M. Stirner 和 G. Seelmann，“JPEG 文件的改进冗余消除”，载于图像编码研讨会论文集，葡萄牙里斯本，2007年11月。[6] D. R. Horn、K. Elkabany、C. Lesniewski-Laas 和 K. Winstein，“一种透明压缩数百拍字节图像文件存储系统的设计、实现与部署”，载于第*14*届 *USENIX 网络系统设计与实现研讨会*（*NSDI*）论文集，美国马萨诸塞州波士顿，2017年3月，第1–15页。[7] M. Stirner，“packMP3 v1.0g”，软件，2016年1月。 [在线]。获取地址：https://github.com/packjpg/packMP3 [8] A. González García，“OptiVorbis v0.3.2”，软件，维戈大学，2026年8月。 [在线]。获取地址：https://github.com/OptiVorbis/ OptiVorbis [9] M. V. Mahoney，“无损数据压缩中上下文模型的自适应加权”，佛罗里达理工学院，美国佛罗里达州墨尔本，技术报告 CS-2005-16，2005年。[10] B. Knoll 和 N. de Freitas，“从机器学习的视角看使用 PAQ8 进行预测编码”，载于数据压缩会议（*DCC*）论文集，美国犹他州斯诺伯德，2012年4月，第377–386页，doi：10.1109/DCC.2012.44。[11] C. Mattern，“数据压缩中的混合策略”，载于数据压缩会议（*DCC*）论文集，美国犹他州斯诺伯德，2012年4月，第337–346页，doi：10.1109/DCC.2012.40。[12] G. N. N. Martin，“区间编码：一种从数字化消息中消除冗余的算法”，载于视频与数据记录会议论文集，英国南安普敦，1979年7月。[13] S. Pfeiffer，“Ogg 封装格式版本 0”，RFC 3533，2003年5月，doi：10.17487/RFC3533。[14] T. B. Terriberry、R. Lee 和 R. Giles，“Opus 音频编解码器的 Ogg 封装”，RFC 7845，2016年4月，doi：10.17487/RFC7845。[15] R. E. Krichevsky 和 V. K. Trofimov，“通用编码的性能”，*IEEE 信息论汇刊*，第27卷，第2期，第199–207页，1981年3月，doi：10.1109/TIT.1981.1056331。[16] I. Pavlov，“LZMA SDK”。访问日期：2026年10月24日。 [在线]。获取地址：https://www.7-zip.org/sdk.html [17] Y. Collet 和 M. Kucherawy，“Zstandard 压缩及‘application/zstd’媒体类型”，RFC 8878，2021年2月，doi：10.17487/RFC8878。[18] M. Mahoney，“ZPAQ v7.15”，软件，2016年8月。 [在线]。获取地址：http://mattmahoney.net/dc/zpaq.html [19] K. Szewczyk，“bzip3 1.5.4”，软件，2026年9月。 [在线]。获取地址：https://github.com/iczelia/bzip3 [20] E. J. Humphrey、S. Durand 和 B. McFee，“OpenMIC-2018：用于多乐器识别的开放数据集”，载于第*19*届国际音乐信息检索学会会议（*ISMIR*）论文集，法国巴黎，2018年9月，第438–444页，doi：10.5281/zenodo.1492445。[21] C. Wang 等，“VoxPopuli：用于表征学习、半监督学习和解释的大规模多语言语音语料库”，载于第*59*届计算语言学协会年会（*ACL-IJCNLP*）论文集，2021年，第993–1003页，doi：10.18653/v1/2021.acl-long.80。[22] V. Pratap、Q. Xu、A. Sriram、G. Synnaeve 和 R. Collobert，“MLS：用于语音研究的大规模多语言数据集”，载于*Interspeech* 论文集，2020年10月，第2757–2761页，doi：10.21437/Interspeech.2020-2826。[23] Mojang，*Minecraft*，电子游戏，瑞典斯德哥尔摩，2011年11月。[24]欧洲广播联盟，“主观测试用音质评估材料录音：EBU SQAM CD 用户手册”，EBU Tech 3253，2008年9月。 [在线]。获取地址：https://tech.ebu.ch/publications/tech3253