无损压缩源代码存储中的能耗–吞吐量权衡

Paolo Ferragina*L'EMbeDS* 系
圣安娜高等研究院比萨，意大利
Paolo.Ferragina@santannapisa.it

Francesco Tosoni，*IEEE* 会员
L'EMbeDS 系 圣安娜高等研究院比萨，
意大利
Francesco.Tosoni@santannapisa.it

arXiv:2601.13220v1 [cs.DS] 19 Jan 2026

摘要—**从大规模源代码归档中检索数据，对于人工智能训练、基于神经网络的软件分析和信息检索等至关重要。本文研究并通过实验探讨了用于索引大规模源代码数据集的压缩键值存储设计，评估其在三种主要计算资源之间的权衡：压缩后的空间占用、时间和能源效率。在国家级高性能计算基础设施上开展的大量实验表明，不同的压缩配置会产生各异的权衡，在实现高压缩比的同时，检索吞吐量和能源效率也提升了一个数量级。我们还研究了数据并行，结果表明，尽管它显著提升了速度，但提高能源效率更为困难，这反映出现代硬件已知的非能耗比例特性，也对时间与能耗直接相关这一假设提出了挑战。本研究简化了能耗感知配置调优的自动化流程，并实现了可部署于持续集成/持续交付流水线的标准化绿色基准测试，从而为系统架构师提供一系列帕累托最优的能耗–压缩–吞吐量权衡方案，以及切实可行的指导方针，帮助构建可持续、高效的存储后端，用于大规模开放源代码归档。**

索引词—**源代码归档、键值存储、无损压缩、绿色软件、大数据管理**

I. 引言

Gemini、ChatGPT、Claude 等大语言模型（LLM）的迅速普及 [19], [38]加剧了获取人类撰写的数字文本的竞争 [6], [43]。由此产生的大规模抓取给维基共享资源 [36] 和 Software Heritage（SWH）等开放档案库带来了压力 [1], [14], [16]。因此，维基媒体报告称，机器人流量占比达到 65%，带宽增长了 50% 2025 [36] ，遗憾的是，这限制了传统用户访问可用的带宽。The Stack v1、The Stack v2 [29], [30] 以及相关的、源自 SWH 的源代码数据集，对于训练 GitHub Copilot [24], [34] 或 Code Llama [39],等代码生成模型至关重要，每月会收到数千次下载请求。SWH 档案库最终决定限制大语言模型批量抓取，这并不令人意外[1]。然而，Winery后端在未排序的 Ceph 数据上进行分块（解）压缩 [44], ，速度仍然太慢。维基媒体也实施了类似限制：最多允许 2 个机器人连接，速率为 25 Mbps[2]。然而，所有这些做法，以及降低文件质量 [47] 或提供付费服务 [20],，都会损害自由访问；它们解决的是表面问题，而非传统基础设施与大语言模型时代当前需求之间的结构性错配。因此，近期的行业举措和欧盟法规将档案带宽限制作为目标，并主张为软件开发提供资金 [17]。能源问题也不容忽视。耗能巨大的数据中心消耗全球电力的 2%，但程序员仍然忽视能源效率 [37]：随着人工智能驱动的磁盘需求增加，存储的隐含碳足迹 [41]也随之恶化。因此，档案可持续性也承担着关键的创新作用 [4]，而（绿色）软件工程师则结合建模与环保技术，以纳入能源相关评估。要应对这些结构性和可持续性挑战，需要能够无缝集成到现代持续集成/持续交付流水线中的自动化、能效感知系统。鉴于 SWH 面临的可扩展性问题，我们设计了一种用于压缩源代码的键值存储，并评估其空间、时间和能耗表现。我们的工作扩展了源代码静态压缩格式，使其不再局限于 MiB/s 级检索 [9],。我们的系统可在单个实例中容纳所有这些数据集，从而扩展至太字节级索引代码库，同时提供动态、能效感知的系统，以相近的压缩大小持续实现 GiB/s 级检索吞吐量。

我们在国家级高性能计算基础设施上的实验表明，存在一些配置能够实现显著压缩，并使吞吐量和能源效率提升一个数量级。我们注意到，数据并行能使完成时间比能源效率更好地随规模扩展，这反映了硬件的非能耗比例特性。因此，我们的研究成果为系统架构师提供了一系列帕累托最优配置和实践，助力构建更绿色、高效的存储后端，以服务于海量开源代码档案库及其他应用。

本研究由阿尔弗雷德·P·斯隆基金会资助，资助编号为 # G-2025-25193 (sloan.org)。© 2026 IEEE。允许个人使用本材料。所有其他用途均须获得 IEEE 许可，无论是在当前还是未来的任何媒体上，包括为广告或宣传目的重印/再版本材料、创作新的合集作品、转售或重新分发至服务器或列表，或在其他作品中重复使用本作品中受版权保护的任何内容。本稿件的正式版本将发表于 SANER 2026 - Greenvolve Workshop 论文集，并可在线查阅：https://doi.org/[TBA]

[1]https://www.softwareheritage.org/legal/bulk-access-terms-of-use[2]https://wikitech.wikimedia.org/wiki/Special:PermaLink/2324939

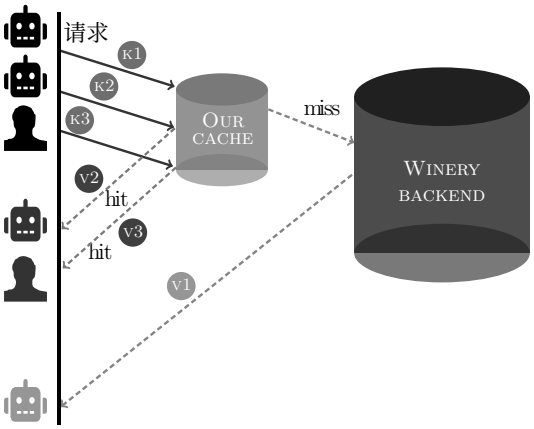


图 1：一种多层存储架构，首先将键查找请求发送到我们的基于PPC的缓存；只有缓存未命中时，才会将查询转发到 Winery 后端。

II. 问题定义与结果

下面将研究场景形式化。给定一个操作流 $\{q_1, \ldots, q_m\}$，其中每个 q_i 都是针对按两级存储方案组织的源代码集合执行的插入、查找、更新或删除操作：该方案由容量为 M 的本地磁盘，以及一个大型基于磁盘的存储 $\mathcal{S}$ 构成。我们的目标是设计一个兼顾时间与能耗性能、容量为 $\leq M$ 的压缩缓存，在支持这些操作的同时，尽可能减少对 $\mathcal{S}$ 的访问。因此，该缓存会为精心筛选出的高频 SWH文件子集建立索引，作为多层对象存储系统（如图 1 所示）中的最后一层；该系统由容量更大但速度更慢的存储支撑，例如托管完整归档的 SWH 基于 Ceph 的 Winery 后端[3] [44], 。这样一来，我们定制的缓存便可适配地理请求模式或爬取机器人，也可为语言及代码到代码检索提供数据泵。我们为无损压缩的 NoSQL 键值存储建立可排序索引键。此键设计以及通过键值存储实现置换-分区-压缩（PPC）范式（见第 III-B 节）的算法思想，是我们提升压缩率、利用文件间冗余并实现太字节规模动态性和可扩展性的基石。

我们将通过探讨三个主要研究问题展开论述：Q1 商用键值存储如何实现 PPC 范式，从而克服 [9], 的规模和静态限制，并在太字节规模下实现稳健的时间、空间和能耗性能？Q2 数据规模、可压缩性与我们所提方案中的时间—能耗权衡之间有何相互作用？Q3 在哪些查询设置和并行配置下，我们的方案能够实现大规模、节能的源代码文件检索？

我们的架构目标是构建一种 SWH 多层存储引擎，如图 1 所示，其大型缓存采用 PPC 范式与商用键值存储的有原则组合。该组合针对特定目标提供不同的空间-时间帕累托解。尽管我们的方案完全动态，但本文仅考察构建阶段（可作为批量或突发插入的代理）和检索阶段；我们不测试删除操作。针对上述三个问题，对于 Q1，我们将设计并分析该方案在多种空间、时间和能耗特性各异的配置中的影响，考察时间-能耗检索、构建效率、压缩比（定义为压缩后数据大小与未压缩数据大小之比：越低越好）以及线程数之间引人关注的权衡。具体而言，我们的键值存储设计存在适当的配置，可将构建的时间-能耗成本降低至原来的3×，并始终将时间-能耗检索性能提升一到两个数量级。对于 Q2，我们将展示，更强的压缩能够节省空间，但会牺牲构建阶段的时间-能耗性能。检索性能则较为复杂，块大小、压缩级别和线程数之间存在复杂的相互作用：使用 zstd-6 和 4 KiB 小块的均衡方案在检索性能上占优（至少可达数 GB/s），但可扩展性仅限于 16–32 个线程。对于 Q3，我们确认，我们的方案在均匀采样键查询下具有显著的可扩展性（检索吞吐量为 8.4–30.0×），但不存在单一的最优方案；相反，存在丰富的帕累托最优谱，使架构师能够根据不同工作负载调整我们的绿色源代码存储系统及类似系统。我们还考察了偏斜查询分布，并报告了在较低并行度下更高的检索吞吐量。尽管如此，我们的方法和结果具有足够的通用性，可扩展至任何冗余文本集合和键值存储架构。

我们将在第 III 节介绍相关概念和背景，随后转向压缩键值存储的设计（第 IV 节）。第 V 节深入介绍代码环境配置，包括高性能计算基础设施（V-A）、数据特征（V-B）以及我们采用的键值存储 RocksDB 的调优（V-C）。在接近结尾时，我们将在第 VI 节报告针对 10-GiB Python 语料库的微基准测试，并在第 VII 节评估大规模多指标性能。最后，第 VIII 节总结我们的研究发现和未来工作。

III. 基本概念

A. Software Heritage

论文 [40]将源代码视为完整的科学与工业人类产物，值得加以保存。正是这种认识催生了软件遗产（SWH）倡议及其基础设施，该倡议由法国国家信息与自动化研究所和联合国教科文组织支持，旨在收集、整理并维护全球最大的开源代码档案库 [14], [16]。SWH 使人们能够追踪软件演化 [15],以监测漏洞 [7],，并为人工智能训练提供自由数据。联合国教科文组织支持 SWH，因为它符合联合国主导的公平人工智能和开放科学议程[4]。CodeCommons 倡议则为此作出贡献

[3]https://docs.softwareheritage.org/devel/swh-objstorage/winery.html

[4]https://unesdoc.unesco.org/ark:/48223/pf0000383771

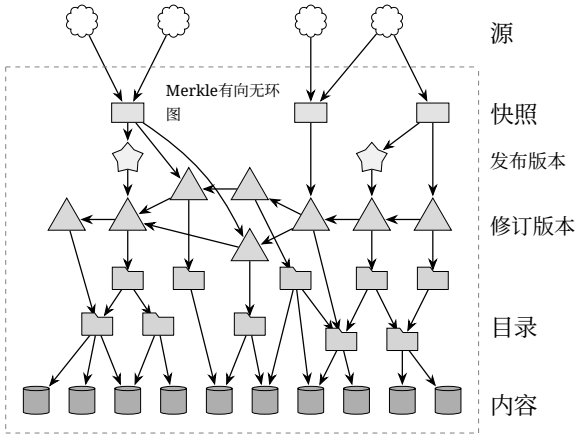


图 2：Software Heritage 数据模型，其中源代码内容构成有向无环图（DAG）的叶节点

SWH 的目标是为适当的 Parquet 格式和人工智能应用精心挑选代码子集并进行分发，例如为 BigCode 的 The Stack 等代码数据集所做的工作 [29], [30]。研究发现 [31]，此类训练代码集能够增强大语言模型的推理能力，甚至包括通用大语言模型。多年来，SWH 持续从 GitHub、GitLab 及众多其他源获取数据，已归档超过 200 亿个文件，数据量超过 2 拍字节，涉及逾 4 亿个项目（2025年10月） [12][5]。这些海量数据带来了基础设施方面的挑战，不仅涉及存储，也涉及索引和检索。图 2 中的代码（元）数据构成一个有向无环图（DAG），其中仅内容（“数据块”）在重复数据删除和压缩后仍占存储空间的 99% [13]。因此，降低当前 SWH 文件系统（SwhFS）的存储成本至关重要[6] 。

B. 置换、分区与压缩

我们的方法基于置换、分区、压缩（PPC）范式 [10], [11], [23], ，这是一种通过将相似数据分组来提升压缩效果的通用策略。据我们所知，尽管 PPC 的适用范围很广，此前唯一一次尝试采用受 PPC 启发的源代码压缩格式，是 [9]，的工作；但其试验仅针对静态代码集合，规模仅达 200 GiB，压缩和检索速度表现平平（通常为 ≈10MB/s），且采用单线程处理，也不支持动态更新。

我们将这项研究拓展到大规模数据集，提出了一种基于键值存储的 PPC 范式新实现，即 RocksDB [18],。我们选择 RocksDB，是因为它具备多线程性能和可扩展性。RocksDB 的日志结构合并树会原生地将数据组织为由排序键构成的压缩块，因此非常适合 PPC 范式；与[9]的方法不同，它还具有支持对索引文件执行动态操作的优势。因此，我们的目标是为第二节中的动态源代码缓存问题提供一种兼顾空间与时间效率、可持续的缓存方案，有效利用 PPC 指导的键设计，将原始数据大小减少约一个数量级，同时提供高效的构建和查找操作。下一节将进一步介绍我们方案的算法见解。

IV. 我们的方案

我们提出一种 PPC 缓存，作为多层存储系统中的最后一层，类似于图 1 所示的系统。我们的缓存为个性化代码库编制索引，以满足用户或应用的特定需求，并根据地理区域、数据爬取机器人的需求以及 CodeCommons 项目中的搜索引擎进行调整 [1]。我们假设，访问频率较低的数据存放在容量更大、速度更慢的对象存储系统中，例如 SWH 的基于 Ceph 的 Winery 后端[7] [44]。因此，我们着手解决基于 RocksDB 构建快速、动态、大规模 PPC 关联存储的问题 [18]。我们选择 RocksDB，是因为 $\mathcal{S}$它是开源软件，具备生产级的稳健性，经过了全面的文献调研 [28, §3],，并支持并行读取且已证明具有良好的可扩展性。它支持所有动态操作，包括更新（通过幂等插入实现）和删除。虽然我们仅关注通过批量插入来构建和检索源代码文件，但我们的设计也支持动态更新和删除，这对于 SWH 的动态特性至关重要。我们的解决方案还对缓存大小施加了限制 M ，容量以太字节计。在通过键值存储实现的 PPC 流水线中，定义用于索引每个键值（源代码）是最关键的设计选择。成功的键分配应将相似的文件内容（值）映射到字典序相近的键，从而通过“线性化分组”实现高效压缩。[9]研究了基于上下文和内容的键设计，得出结论：文件名在压缩比与计算速度之间实现了最佳权衡。因此，我们采用指定的文件名作为键，具体方法如下：对于 SWH 档案库的默克尔树中与每段内容相关联的多个文件名，[15]所述方法使用“热门内容文件名数据集”来确定出现频率最高的文件名，并将其作为规范标识符。随后，我们通过将文件扩展名置于文件名前面来构造基于文件名的键（例如 h.doxygen），再附加一个唯一的 SWH 标识符（即所谓的 SWHID[8]），以确保每个键都不重复。这样，我们先按扩展名（作为语言的代理）对文件分组，再按文件名分组，确保相似文件在 RocksDB 序列化布局和压缩块中相邻。

V. 代码设置

A. 高性能计算基础设施

我们在 Kraken 集群的 CPU 分区上进行了所有实验。该集群是法国国家级高性能计算与数据分析 (HPCDA) 基础设施，由 Grenoble Alpes Recherche/GRICAD 运营。我们利用 CPU 集群，在 C++/CMake 并行流代码中实现多核并发，该代码基于标准 POSIX 线程（即 pthreads）实现。

[5]https://unesdoc.unesco.org/ark:/482230000371017 6 / pf https://docs.softwareheritage.org/devel/swh-fuse

[7]https://docs.softwareheritage.org/devel/swh-objstorage/winery.html [8] https://www.swhid.org

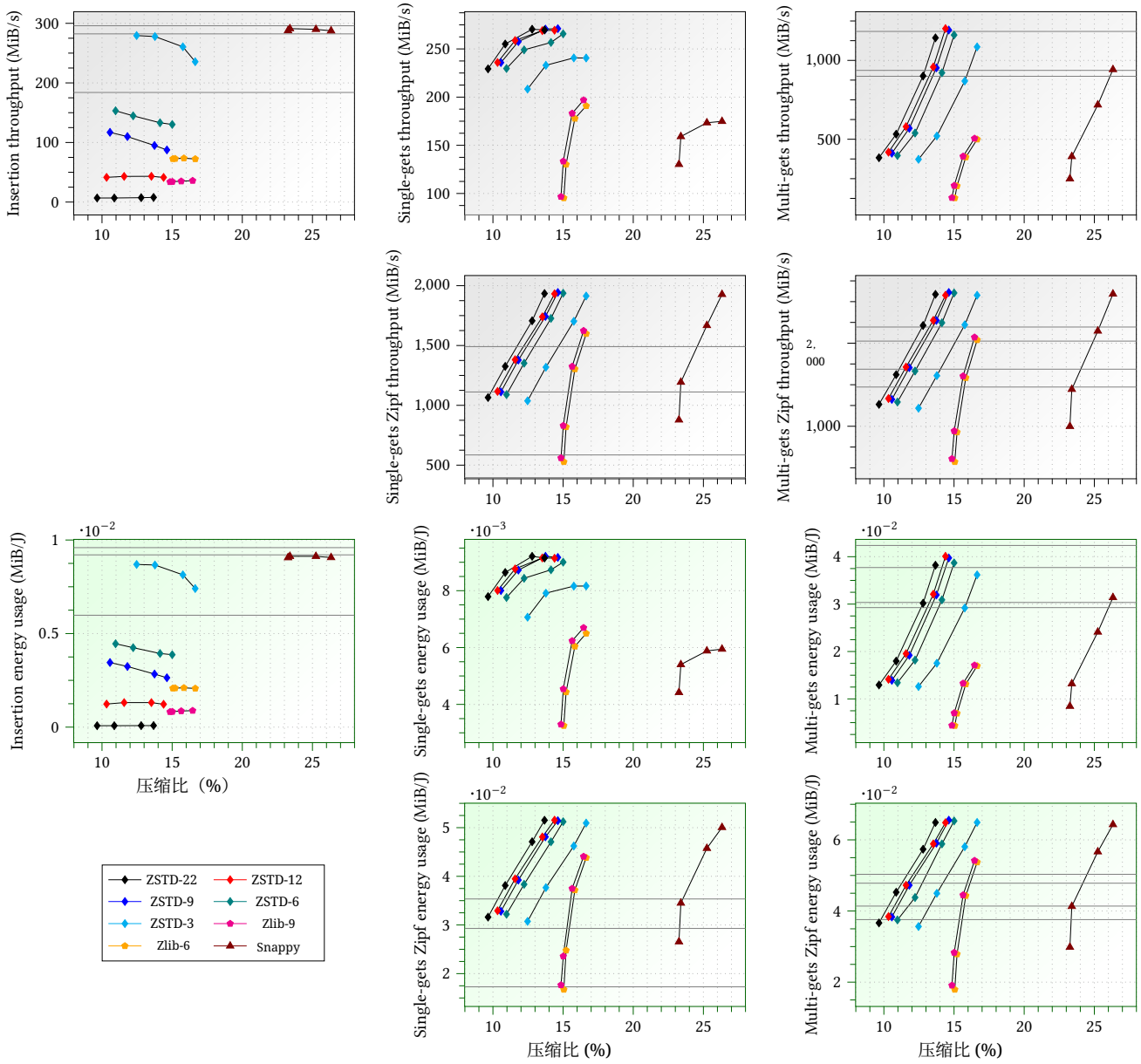


图 3：我们针对 10 GiB Python 代码进行的微基准测试。 y 轴表示吞吐量（时间以 MiB/s 表示，能耗以 MB/J 表示）； x 轴显示压缩比。每个图中，性能最佳的系统配置位于左上角。

CPU 分区包含多种节点类型，它们均配备相同的处理器（2× AMD EPYC 9654 96C 2.4 GHz，共 192 个核心）、本地存储（1.92 TB NVMe 固态硬盘）以及 200 Gb/s InfiniBand NDR HCA 互连。我们在 44 个配备 768 GB 随机存取存储器的标准节点上运行缓存。集群工作负载管理由 OAR v3 批处理调度器负责。为确保环境条件一致且可复现，我们始终通过 -p "fat = 'NO'" 将启动的作业限制在标准节点上。我们将输入数据存储在 Bettik[9], , 这是一个高性能共享暂存存储，采用 BeeGFS 文件系统，具有低延迟访问能力，并持续传输速率最高可达 10 Gb/s。我们将之后所有 RocksDB 序列化数据存储在 Hoyt[10],上。Hoyt 是一套基于全闪存基础设施构建的高性能分布式暂存系统，配备 NVMe 控制器。它采用 BeeGFS，并通过支持远程直接内存访问（RDMA）的 200 Gb/s InfiniBand 网络连接，可提供低延迟、高吞吐量的输入/输出，适用于输入/输出密集型和并行工作负载。

鉴于不可能为每个集群节点都连接功率计，我们使用 Linux 性能分析器 perf 进行能耗估算。在我们的 AMD 基础设施上，我们依靠系统通过其性能

[9]https://gricad-doc.univ-grenoble-alpes.fr/hpc/data management/bettik

[10]https://gricad-doc.univ-grenoble-alpes.fr/hpc/data management/silenus

计数器报告能耗数据的能力，这项功能类似于英特尔的 RAPL 接口，后者通过模型专用寄存器（MSR） [26]运行。我们在封装级别进行测量，旨在捕获整个处理器插槽的功耗，展示软件工程师如何利用现成的性能分析工具，在无需专用硬件的情况下做出能耗感知型决策。众所周知，类似 RAPL 的接口存在误差范围，但我们认为，相对配置比较仍然有效，例如 [26], [42]中采用的 Perf。

B. 数据集

在我们的实验中，所有输入数据均采用 Parquet 列式格式，并使用 Snappy 压缩；这是 Pandas 和 PyArrow 等库的默认设置。

语言	大小（GiB）	文件数	文件大小（千字节）	
			Mean	中位数
Python	200.0	9.640.731	21.7	7.2
C/C++	200.0	6.437.613	32.5	9.0
JavaScript	200.0	3.464.374	60.5	4.6
Java	200.0	26.373.974	7.9	2.4

表 I：Boffa等人 [9]的数据集特征。

我们考察了四种主要语言的源代码：Python、Java、JavaScript 和 C/C++。原始的 200 GiB tar 文件被转换为 Parquet，并划分为每组 100,000 行的行组，以平衡 I/O 效率和内存使用。较大的行组能提高压缩率并减少元数据开销，但会增加内存需求。该数据集的特征此前曾被 [9] 用于分析 SWH 内部及文件间的相似性，详情见表 I。在对大型数据进行索引（第 VII 节）之前，我们先针对 434,322 个 Python 文件（10 GiB）进行了微基准测试（第 VI 节）。我们选择 Python，是因为 [9] 表明它具有很高的可压缩性，为评估压缩索引能力提供了理想条件，也便于与先前工作直接比较。

C. RocksDB 配置

为确保实验可复现，我们针对硬件和一次写入、多次读取工作负载配置了 RocksDB，并利用 768 GiB 的节点 RAM 来提升速度、节省空间和降低能耗。相较于默认设置，主要调整包括启用内存映射 I/O（allow_mmap_reads、allow_mmap_writes），增大写缓冲区（write_buffer_size 设为 2 GiB），并针对并发（use_adaptive_mutex、六个压缩线程）和读取密集型访问（optimize_filters_for_hits）进行优化。我们关闭了预写日志（WAL）压缩（wal_compression），并将 WAL 总大小设为 64 GiB，以优先降低写入延迟。后续所有关于压缩器选择、块大小和太字节级性能的实验均使用此配置。

VI. 在 10 GIB 的 Python 语料库上进行微基准测试

我们使用包含 434,322 个 Python 文件、总计 10 GiB 的语料库，建立了单线程性能基线。我们测试了构建插入，以及按原始（排序后）的数据集顺序访问键的单键和多键检索；对于多重获取，我们每批使用 100 个键。我们还评估了幂律键分布（$\alpha = -1.5$），以模拟现实场景：用户会并发查询相似项目；这有助于模拟频繁键或术语的最佳情况热启动 [33, §5.1.2]或网络搜索 [2], [32]。我们对五次执行的能耗测量结果取平均值。图 3 展示了结果。上方子图（灰色背景）显示数据吞吐量（MB/s）；下方子图（绿色背景）显示能源效率（MB/J）。各图中，左上角表示性能最佳的配置。标记形状表示压缩算法，颜色表示压缩级别；每条系列对应 4、16、64 和 128 千字节的块大小，从右向左排列，块大小越大，压缩效果越好。灰线表示未压缩基线。图中显示时间—能耗相关性很强，因此下文仅讨论上方的图。显然存在一种权衡：增大块大小可以提升压缩效果，对插入吞吐量影响不大，却会减慢检索速度；这分别体现在插入时从右向左的水平趋势，以及检索时从上到下的垂直趋势。提高 zstd 和 zlib 的压缩级别会提升插入性能，但会降低单次获取的检索性能。zstd 在两项指标上均优于 zlib，因此我们在后续实验中不再考虑 zlib。我们也排除了 Snappy，因为它的插入性能仅略优于 zstd，存储占用却约为其两倍，且检索性能更差。因此，我们专注于 zstd。在级别 9、12 和 22 下，其检索性能相近，但插入性能差异明显。zstd-22 的插入吞吐量峰值为 7.8 MiB/s；要为第 V-B 节中的 800 GiB 建立索引，这将需要≈29 小时。zstd-12 速度更快，但其插入吞吐量仍比 zstd-9 慢超过 $2\times$，而单次获取仅提升 0.6%，压缩比也相近。级别 3 和 6 的插入速度更快，但压缩效果（最多降低 2%）和检索性能（相对降低。4% 至。9%）略有下降。至于其余的多重获取和幂律分布图，我们观察到类似的优势趋势，且速度更快——幂律多重获取的计算速度快一个数量级以上。根据这项微基准分析，我们采用四种大规模配置：

- 采用 64 KiB 块的 zstd-3，在插入方面优于较小块，同时检索性能也具有竞争力。
- 采用 4 KiB 块的 zstd-6 可最大限度地提升单次获取速度；采用 128 KiB 块时，其压缩效果优于 zstd-3，检索性能则相近。
- 采用 128 KiB 块的 zstd-9，在压缩效果方面优于 zstd-6，同时插入和检索性能相当。

VII. 扩展至大型集合

尽管我们的缓存是完全动态的，但本文评估了构建过程（VII-A）以及并行单次获取和多次获取（VII-B）。

A. 构建阶段

根据第 VI 节中的考量，我们构建了四个独立的 RocksDB 实例，每个实例对应一种

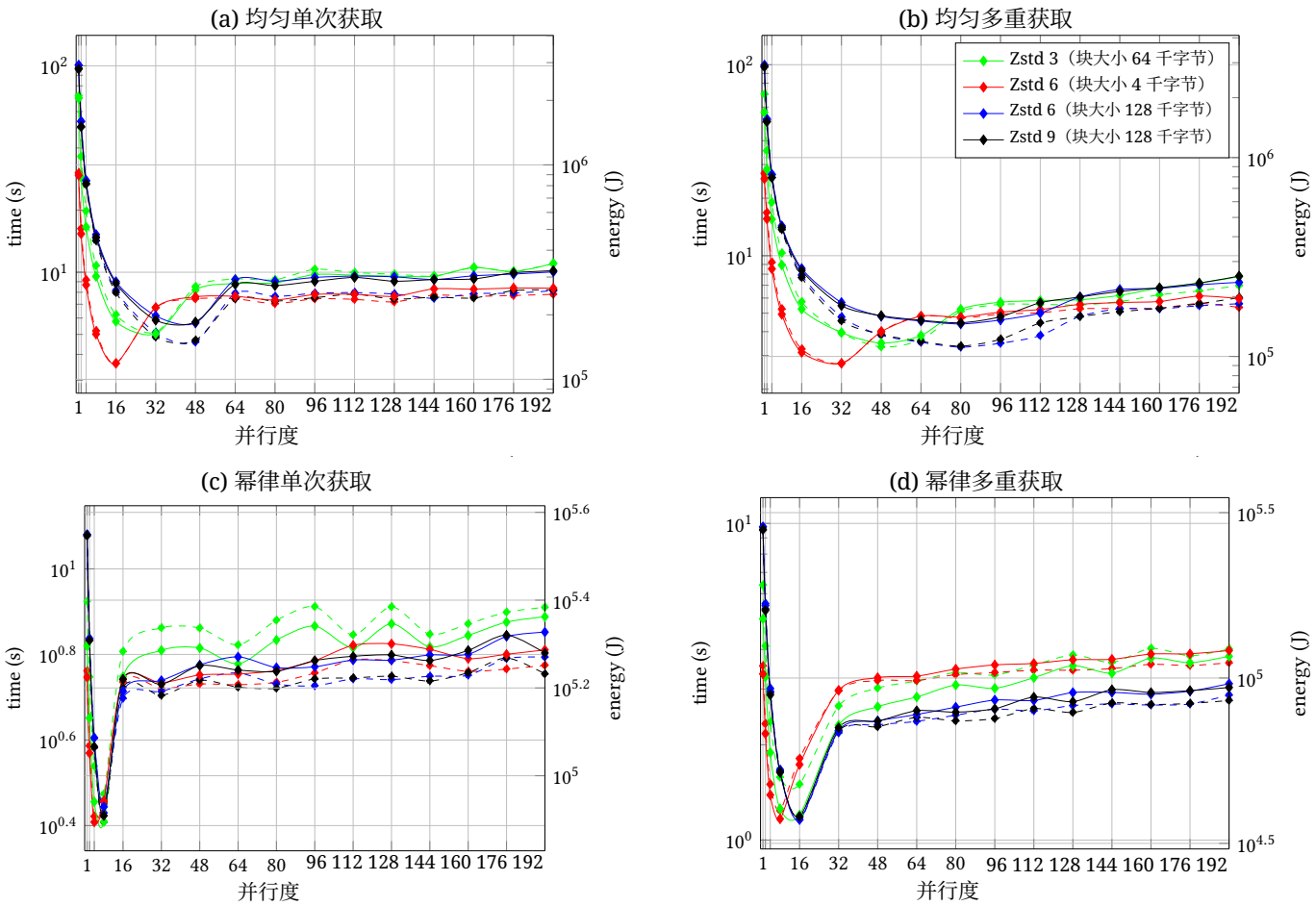


图4：查询表 I 中数据所需的时间（虚线）和能耗（实线）。

压缩器配置：zstd-3（块大小为 64 千字节）、zstd-6（4 和 128 千字节），以及 zstd-9（128 千字节）。我们报告了对代码数据进行索引和查询的实验结果，这些数据来自一个太字节规模的数据集，该数据集由第V-B节中的四个 200 GiB 语言专属源代码 Parquet文件合并而成。Parquet 数据流按键排序，因此可在连续的 RocksDB SSTable文件中实现高效索引；这类顺序可预测的访问有助于高效预取并减少缓存未命中。双层I/O模型 [22,§4.2] [3]确实阐明了随机访问会导致存储层次结构低效；相比之下，像缓存无关算法 [21] [22,§2], 中的访问局部性则可以缓解这些问题。

压缩	Time	能耗	压缩比率（%）	吞吐量（MiB/s）
zstd-3（64 千字节）	1小时 1分钟 21秒 3681秒	1086 千焦 0.30 千瓦时	19.05	446.67
zstd-6（4 千字节）	2小时24分2秒 8642秒	2518 千焦 0.70 千瓦时	19.85	190.28
zstd-6（128 千字节）	2小时7分11秒 7631秒	2232 千焦 0.62 千瓦时	15.98	215.49
zstd-9（128 千字节）	2小时53分44秒 10424秒	3047 千焦 0.85 千瓦时	15.49	157.75

表 II：表 I 中数据的构建指标。

表 II 报告了构建时间、能耗和插入吞吐量，所有配置均实现了较高的吞吐量（158–447 MB/s）。zstd-9（128 KiB）压缩效果最佳，但构建速度最慢、能耗最高。相比之下，zstd-3（64 KiB）的压缩比更低，却是速度最快、能效最高的配置。较低的压缩级别可优化速度和能耗，而较高级别则会带来显著成本，压缩收益却很有限。我们的压缩比为 15–20%。我们发现，块大小与性能之间存在复杂的相互作用：采用 4 KiB 块的 zstd-6 速度比使用 128 KiB 数据块时更慢，这表明小块开销可能超过压缩收益。能耗会随着压缩激进程度的提高而增加。zstd-9（128 KiB）消耗了 ≈0.85 kWh，足以为一辆普通欧洲电动汽车提供 ≈4 km 的行驶电量 [45]，或满足美国住户每日能耗的 ≈ 3% [25]。相比之下，zstd-3（4 KiB）仅需 0.30 kWh（约可供电动汽车行驶 1.4 km）。我们注意到，吞吐量与能源效率高度相关，但并非成正比。

B. 多线程检索

我们使用命中查询评估检索，因为未命中 可由布隆过滤器高效处理 [8]。我们将并行吞吐量与最佳串行基线进行比较。为考察随机访问性能，我们抽取一百万个互不相同的随机键，这可能迫使每次查询都解压一个不同的数据块。这样，我们模拟了对内存层次结构的低效利用，与 RocksDB 的点查找评估相符 [18]。我们还测试幂律选择（$\alpha = -1.5$），其中重复键有利于访问局部性。对于多重获取，所有情况下每次查询均获取 100 个键值对。大小为 p 的线程池使用 std::future 和 std::async 处理查询，遵循现代 C++ 并发实践 [46, §4.2]。图 4 展示了表 I 中四种语言的流并行检索，线程数从 1 到 192 不等。对于均匀键分布，随着线程数增加，我们观察到性能始终提升 8.4× 至 30.0× 倍；能耗可扩展性与之高度相关，但提升幅度较小，这反映了多核处理器的能耗并非与性能成正比 [27], [42]。对于均匀查询分布，存在多种帕累托最优配置。zstd-6（4 KB）速度最快，单次获取的最优线程数为 16，多重获取则为 32。zstd-6（128 KiB）位居第二，而 zstd-3 以较少的线程数即可达到相近速度。zstd-9（128 KiB）最慢，且需要更高的最优线程数（>80）。对于幂律单次获取，情况则恰好相反：低并行度下（4 个线程），zstd-6（4 KiB）最快。其他所有配置都需要更高的并行度才能达到相近性能。幂律查询比均匀查询更快，但可扩展性有限，性能平台期与阿姆达尔定律相符 [5],[35, §2.5.4]。对于均匀分布，加速比介于 8.4 至 22.0× （单次获取）和 9.9 至 30.0× （多重获取）之间，因此运行时间始终缩短了一个到两个数量级。对于幂律情况，两者分别达到 2.2 至 4.7× 和 3.0 至 5.4×。能耗方面也取得了类似的收益。图 5 根据图 4 展示了构建和均匀单次获取的时空帕累托最优配置。图例标出了颜色对应的压缩器及其等级、块大小，以及单次获取的最优线程数。就构建而言，zstd-3（64 KiB）、zstd-6（128 KiB）和 zstd-9（128 KiB）均位于帕累托前沿，代表了不同的速度—压缩权衡。

zstd-6（4 千字节）虽然处于劣势，但其最佳配置所需的线程数较少（32），因此每个 192 核节点上可运行多个相互隔离的 RocksDB 实例，用于镜像、冗余或多租户场景。检索速度比插入快一个数量级，这也符合 LSM 树读写不对称的特点 [28, §3] （写入需要进行压缩整理）。对于单次检索，zstd-6（4 千字节）位于帕累托前沿，是时间和能耗表现最佳的检索算法，尽管其空间效率较低。我们的实现有许多方面优于 [9]：它支持动态处理，在更大的数据集上实现了相近的压缩率（15–20%），检索速度也始终达到 GiB/s，而对方的静态方案为 ≈10MiB/s。我们将 Perf 测得的能耗降低了一个至两个数量级，朝着高

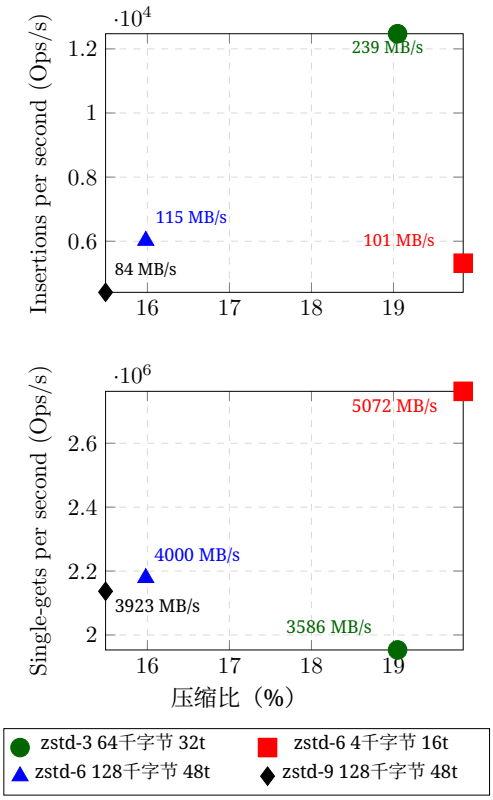


图 5：表 I 中四个合并数据集的最佳配置比较。

适用于动态、大规模环境的性能解决方案。

VIII. 结论与未来工作

在大语言模型时代对大规模、可持续基础设施的需求推动下，本文分析并实验研究了一种基于知名数据库 RocksDB[18]、用于索引大规模源代码数据集的压缩键值存储设计。我们从空间、时间和能耗性能三个方面对 RocksDB 进行了评估，结果表明，微基准测试和适当调优能够带来多种帕累托最优解，在时间-能耗检索、构建效率、压缩比和线程数等方面实现颇具吸引力的权衡。极端压缩级别（如 zstd-3 和 ZSTD-22）在提升特定计算资源利用率方面表现出色，而 zstd-6 和 zstd-9 等均衡级别通常能提供最稳健的方案。未来，我们将通过在不同节点上运行多个 RocksDB 实例，并按编程语言对源代码文件进行分区，将分析扩展至分布式集群。我们还将探索集成由大语言模型设计的键，以进一步提升压缩性能。最后，我们将测试把本方案集成到 SWH 采用的开源 Winery 基础设施中的效果。

致谢与可用性

本文所介绍的所有计算均使用 GRICAD 基础设施（https://gricad.univ-greno

ble-alpes.fr）完成，该基础设施由格勒诺布尔的研究团体提供支持。用于复现实验的完整代码库可在 https://github.com/ftosoni/green-compressed-s torage 获取。

参考文献

[1] Mathieu Acher。生成式人工智能赋能生成式编程：自动化代码变体并探索大语言模型的边界，2024 年 11 月。[2] Lada A. Adamic 和 Bernardo A. Huberman。万维网的幂律分布。*Science*，287(5461):2115–2115，2000 年。[3] Alok Aggarwal 和 S. Vitter, Jeffrey。排序及相关问题的输入输出复杂度。*Commun. ACM*，31(9):1116–1127，1988 年 9 月。[4] Anastasia Ailamaki 等人。数据库研究的剑桥报告，2025 年。[5] Gene M. Amdahl。论采用单处理器方法实现大规模计算能力的有效性。载于 *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*，AFIPS '67（春季），第 483–485 页，美国纽约州纽约市，1967 年。美国计算机协会。[6] Stefan Baack、Stella Biderman、Aviya Skowron 和 Kasia Odrozek。迈向大语言模型训练开放数据集的最佳实践，2025 年。访问日期：2025-05-07。[7] Olivier Barais、Roberto Di Cosmo、Ludovic M´e、Stefano Zacchiroli 和 Olivier Zendra。面向网络安全的软件识别：面向监管机构的调查与建议。技术报告，Software Heritage Security 项目（SWHSec），2025 年 3 月。[8] Burton H. Bloom。允许误差条件下哈希编码中的空间/时间权衡。*Commun. ACM*，13(7):422–426，1970 年 7 月。[9] Antonio Boffa 等人。大规模源代码数据集的可压缩性。*Journal of Systems and Software*，227:112429，2025 年。[10] Adam L. Buchsbaum、Donald F. Caldwell、Kenneth Ward Church、Glenn S. Fowler 和 S. Muthukrishnan。海量表的压缩工程：一种实验性方法。载于 *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*，SODA，第 175–184 页，2000 年。[11] Adam L. Buchsbaum、Glenn S. Fowler 和 Raffaele Giancarlo。利用组合优化改进表压缩。*J. ACM*，50(6):825–851，2003 年 11 月。[12] Roberto Di Cosmo、Carlo Montangero、Guido Scatena 和 Laura Bussi。利用 Software Heritage 获取流程保存标志性遗留软件。载于 Zijun Chen（编），*Proceedings of the 17thInternational Conference on Digital Preservation, iPRES 2021, Beijing, China, October 19-22, 2021*，2021 年。[13] Roberto Di Cosmo 和 Stefano Zacchiroli。Software Heritage：为何以及如何保存软件源代码。载于 *iPRES 2017: 14th International Conference on Digital Preservation*，日本京都，2017 年。[14] Ludovic Court`es、Timothy Sample、Stefano Zacchiroli 和 Simon Tournier。源代码归档助力可复现部署。载于 *Proceedings of the 2nd ACM Conference on Reproducibilityand Replicability*，ACM REP '24，第 36–45 页，美国纽约州纽约市，2024 年。美国计算机协会。[15] Ad`ele Desmazi`eres、Roberto Di Cosmo 和 Valentin Lorentz。透过 Software Heritage 的视角回顾 50 年编程语言演化。载于 *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*，第 372–383 页，2025 年。[16] Roberto Di Cosmo。使用 Software Heritage 归档和引用源代码。载于 Anna Maria Bigatti、Jacques Carette、James H. Davenport、Michael Joswig 和 Timo de Wolff（编），*Mathematical Software – ICMS 2020*，第 362–373 页，Cham，2020 年。Springer International Publishing。[17] Roberto Di Cosmo 等人。别再把代码当作事后之物：记录、共享并重视代码。*Nature*，646:284–286，2025 年。评论。[18] Siyi ng Dong、Andrew Kryczka、Yanqin Jin 和 Michael Stumm。RocksDB：面向服务于大规模应用的键值存储开发优先级的演化。*ACM Trans. Storage*，17(4)，2021 年 10 月。[19] Murillo Edson de Carvalho Souza 和 Li Weigang。Grok、Gemini、ChatGPT 和 DeepSeek：对话式人工智能的比较与应用，2025 年 2 月。[20] Lieven Eeckhout。计算领域的可持续性鸿沟：何去何从？*Commun. ACM*，68(3):70–79，2025 年 2 月。[21] Rolf Fagerberg。《缓存无关模型》。载于 Ming-Yang Kao（编），《算法百科全书》，第264–9页。Springer，2016年。[22] Paolo Ferragina。《算法工程学珍珠》。剑桥大学出版社，2023年。[23] Paolo Ferragina 和 Giovanni Manzini。关于文本网络的压缩。载于《第三届 ACM国际网络搜索与数据挖掘会议论文集》（WSDM '10），第391–400页，美国纽约州纽约市，2010年。美国计算机协会。[24] Saki Imai。GitHub Copilot 能否取代人类结对编程？一项实证研究。载于《ACM/IEEE第44届软件工程国际会议：伴随论文集》（ICSE '22），第319–321页，美国纽约州纽约市，2022年。美国计算机协会。[25] Elham Jahani、Kristen Cetin 和 In Ho Cho。利用基于遗传算法的数值矩匹配技术预测城市尺度的独栋住宅建筑能耗。《建筑与环境》，172:106667，2020年。[26] Kashif Nizam Khan、Mikael Hirki、Tapio Niemi、Jukka K. Nurminen 和 Zhonghong Ou。RAPL 实践：使用 RAPL 进行功率测量的经验。《ACM建模、性能评估与计算系统汇刊》，3(2)，2018年3月。[27] Semyon Khokhriakov、Ravi Reddy Manumachu 和 Alexey Lastovetsky。多核处理器计算并非能耗成比例：实现能耗与性能双目标优化的机遇。《应用能源》，268:114957，2020年。[28] Martin Kleppmann。《设计数据密集型应用》。O' Reilly Media，第1版，2017年4月。[29] Denis Kocetkov 等。《The Stack：3 TB 宽松许可源代码》。预印本，2022年。[30] Anton Lozhkov 等。StarCoder 2 与 The Stack v2：新一代，2024年。[31] Aman Madaan、Shuyan Zhou、Uri Alon、Yiming Yang 和 Graham Neubig。代码语言模型是少样本常识学习者。载于 Yoav Goldberg、Zornitsa Kozareva 和 Yue Zhang（编），《2022年自然语言处理实证方法会议论文集》，第1384–1403页，阿布扎比，阿拉伯联合酋长国，2022年12月。计算语言学协会。[32] Aniket Mahanti、Niklas Carlsson、Anirban Mahanti、Martin Arlitt 和 Carey Williamson。长尾的故事：互联网测量中的幂律。《IEEE网络》，27(1):59–64，2013年。[33] Christopher D. Manning、Prabhakar Raghavan 和 Hinrich Sch¨utze。《信息检索导论》。剑桥大学出版社，2008年。[34] Antonio Mastropaolo 等。代码生成技术的稳健性：针对 GitHub Copilot 的一项实证研究。载于《2023年IEEE/ACM第45届软件工程国际会议》（ICSE），第2149–2160页，2023年。[35] Michael McCool、James Reinders 和 Arch Robison。《结构化并行编程：高效计算模式》。Morgan Kaufmann Publishers Inc.，美国加利福尼亚州旧金山，第1版，2012年。[36] Birgit Mueller、Chris Danis 和 Giuseppe Lavagetto。爬虫如何影响维基媒体项目的运行。Diff，2025年4月。[37] Candy Pang、Abram Hindle、Bram Adams 和 Ahmed E. Hassan。程序员对软件能耗了解多少？《IEEE Software》，33(3):83–89，2016年。[38] Areeg Fahad Rasheed、Shimam Amer Chasib、Huda Najm Alabbas、M. Zarkoosh 和 Safa F. Abbas。使用大语言模型评估雅思写作任务：ChatGPT、Claude、Copilot 和 Google Gemini 的比较研究。载于《2024年IEEE机器学习与应用网络技术国际会议》（ICMLANT），第32–37页，2024年。[39] Baptiste Rozi`ere 等。Code Llama：面向代码的开放基础模型，2024年。[40] Len Shustek。我们应该收集什么来保存软件史？。《IEEE计算史年鉴》，28 112, 110 04 111, 2006 ( ): – October 。[41] Swamit Tannu 和 Prashant J. Nair。固态硬盘的肮脏秘密：隐含碳。《SIGENERGY能源信息评论》，3(3):4–9，2023年10月。[42] Francesco Tosoni、Philip Bille、Valerio Brunacci、Alessio de Angelis、Paolo Ferragina 和 Giovanni Manzini。通过无损压缩格式实现更绿色的矩阵运算。《IEEE Access》，13 56779, 2025 56756: – 。[43] Hugo Touvron 等。Llama：开放且高效的基础语言模型，2023年。

[44] Sage A. Weil、Scott A. Brandt、Ethan L. Miller、Darrell D. E. Long 和 Carlos Maltzahn。Ceph：一种可扩展、高性能的分布式文件系统。载于第七届操作系统设计与实现研讨会论文集，OSDI '06，第307–320页，美国，2006年。USENIX 协会。[45] Martin Weiss、Trey Winbush、Alexandra Newman 和 Eckard Helmers。欧洲电动汽车能耗。《可持续性》，16(17)，2024年。[46] Anthony Williams。*C*++ 并发实战。Manning Publ.，Shelter Island，纽约州，2012年。[47] Aviad Zuck、Donald Porter 和 Dan Tsafrir。通过数据降级拯救地球。载于第十九届操作系统热点研讨会论文集，HotOS '23，第61–69页，美国纽约州纽约市，2023年。美国计算机协会。