

LLCBench: Benchmarking Lossless Compression of Large Language Models

Hui Sun^{1,2}, Jiashun Chen³, Huidong Ma^{1,2}, Xiantong Luo⁴,
Chang Xu⁵, Haonan Xie⁶, Lizheng Zu⁷, Qi Ming⁴, Xiaoguang Liu¹,
Gang Wang¹, Cheng Zhong⁸, Wentong Cai¹

¹College of Computer Science, Nankai University, Tianjin 300350, China;

²College of Computing and Data Science, Nanyang Technological University, Singapore 639798;

³School of Software Engineering, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China;

⁴College of Computer Science, Beijing University of Technology, Beijing 100124, China;

⁵Microsoft Research Asia, Beijing 100080, China;

⁶Huanjiang Laboratory, Zhejiang University, Zhuji, Zhejiang 311800, China;

⁷College of Electronics and Information Engineering, Shenzhen University, Shenzhen 518060, China;

⁸School of Computer, Electronics and Information, Guangxi University, Nanning 530004, China

Abstract

The rapid proliferation of Large Language Models (LLMs) has posed significant challenges to model storage, distribution, and inference. While various reviews and benchmarks for lossy compression like distillation, pruning, and quantization have emerged to address these issues, their inherent lack of data integrity restricts their utility in scenarios requiring high-fidelity persistent backup and the reliable sharing of LLM repositories. To this end, we present the first review and benchmark of lossless compression for LLMs, serving as a further complement to existing lossy work. Specifically, we first propose a classification methodology for LLM lossless compression, including Floating-Point-Driven (FPMD), Large-Language-Model-Driven (LLMD), Learning-Based Lossless Compression (LLC), and Traditional Lossless Compression (TLC). Following this, we conduct a detailed review of these four types of methods and discuss their advantages and shortcomings. Subsequently, we evaluate the lossless compression performance, throughput, and resource consumption of 24 solutions across 10 real-world LLMs using our benchmark. This comprehensive evaluation covers 10 critical metrics and spans over 5,700 hours of CPU/GPU computation time. Furthermore, drawing upon our systematic review and benchmark analysis, we identify several promising future research directions, including automated parameter optimization, intelligent storage and distribution frameworks, parallel acceleration for compression algorithms, and the development of robust algorithmic designs. We organize all results and scripts as the LLMs Lossless Compression Benchmark (LLCBench), which can be found at <https://github.com/fahaihi/LLCBench>.

Keywords: Lossless compression; large language models; benchmark; data compression; big data management.

1 Introduction

The widespread application of Large Language Models (LLMs) has fueled remarkable growth in areas such as embodied intelligence [1; 2; 3], AI for science [4; 5; 6; 7; 8], and financial technology [9; 10; 11] et al. However, the extensive parameter scale of LLMs limits their storage, sharing, and management. For instance, the DeepSeek-V3.2 ¹ as shown in Fig. 1(b) features 671 B (billion) parameters, while the latest Kimi-K2.5 model ² reaches a scale of 1 T (trillion) parameters. The enormous parameter scale of these models leads to prohibitive storage and transmission overheads. For example, Tables 1(b) and 1(c) present the storage costs and cloud server egress costs of the top 10 most popular models on Hugging Face ³. Taking the DeepSeek-R1 [12] as an example, its costs are \$69,362,695 for storage and \$12,914,613 for egress traffic globally ⁴.

¹<https://huggingface.co/deepseek-ai/DeepSeek-V3.2-Exp>

²<https://huggingface.co/moonshotai/Kimi-K2.5>

³<https://huggingface.co/models?sort=likes> (until 12, 2025)

⁴The storage and network egress costs are calculated by multiplying the number of likes or downloads by the model size and then by the cost per terabyte (TB). Storage and network egress costs are respectively based on Samsung's \$60 / TB rate and Google's

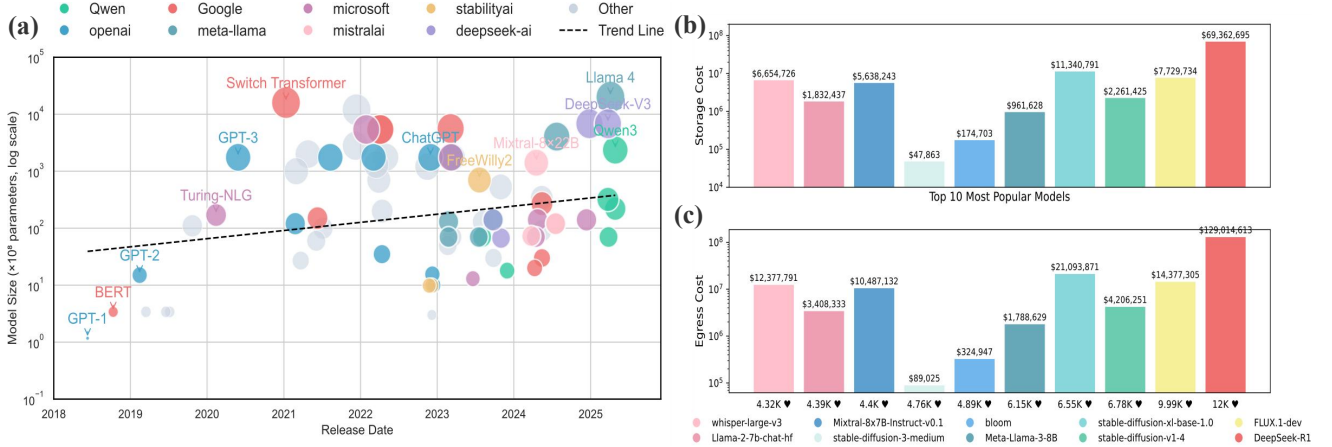


Figure 1: (a) The scale growth trend of LLMs in the Hugging Face model hub, (b) Storage cost of the top 10 most popular models on Hugging Face and (c) Egress cost of top 10 most popular models on Hugging Face as of November 2025.

Although lossy compression techniques, including pruning [13; 14], quantization [15; 16; 17], and distillation [18; 19; 20; 21], have been introduced to reduce the parameter scale of LLMs and enhance inference efficiency, these approaches inevitably compromise data integrity. Consequently, they are inadequate for tasks requiring robust storage, reliable backup, and extensive distribution of LLMs. Fortunately, lossless compression addresses the inherent limitations of lossy methods while offering the following advantages: 1) Reducing the data size of LLMs, thereby lowering the cost of constructing fundamental storage infrastructure. 2) Minimizing data transmission scale, which enhances throughput efficiency during transfer in the web-word. 3) Facilitating downstream applications, for example, ZipNN [22], FM-Delta [23] and Git-theta [24] can be used to optimize Gradients and Optimizers, as well as incremental compression for checkpoint checks.

Recently, some specialized compression schemes targeting multi-modal data [25; 26; 27], floating-point numbers [28; 29; 30; 31], and LLMs-dedicated [22; 23; 24] have been proposed for LLM lossless compression. However, choosing an appropriate compression strategy for practical engineering applications is challenging. For instance: 1) Checkpoint backups scenarios and long-term storage demand extreme compression ratios. 2) Instant transmission in model repositories requires a balance between compression ratio and throughput. 3) Devices with limited memory resources necessitate algorithms with greater operational robustness.

As far as we know, some lossless compression reviews and benchmarks targeting multi-model data [27; 32; 31; 33; 34] and specific domains [35; 36; 37; 38] have already been proposed, but research on LLMs has not yet been conducted. To this end, we propose the first review and LLMs Lossless Compression Benchmark (LLCBench), involving the following contributions:

- We comprehensively review the latest advancements in lossless compression techniques for LLMs, conduct an in-depth analysis of their strengths and limitations, and introduce a systematic classification framework, categorizing them into Floating-Point-Driven (FPMD), Large-Language-Model-Driven (LLMD), Learning-based Lossless Compression (LLC), and Traditional Lossless Compression (TLC).
- We comprehensively evaluate 28 (including 3 FPMD, 2 LLMD, 5 LLC and 14 TLC methods) lossless compressors across 10 widely shared LLMs, like GPT, BERT, Qwen series, assessing multiple metrics including compression ratio, compression and decompression time, resource consumption, and robustness, thereby quantifying their potential differences.
- We highlight the potential research directions for lossless compression of LLMs, including automated parameter tuning, smart storage and distribution, compression parallel acceleration, and robust algorithm design.

The remaining sections are organized as follows: The section 2 introduces related work, the section 3 classifies and reviews lossless compression methods for LLMs, the section 4 covers benchmark testing, the section 5 explores future directions, and the section 6 summarizes the paper.

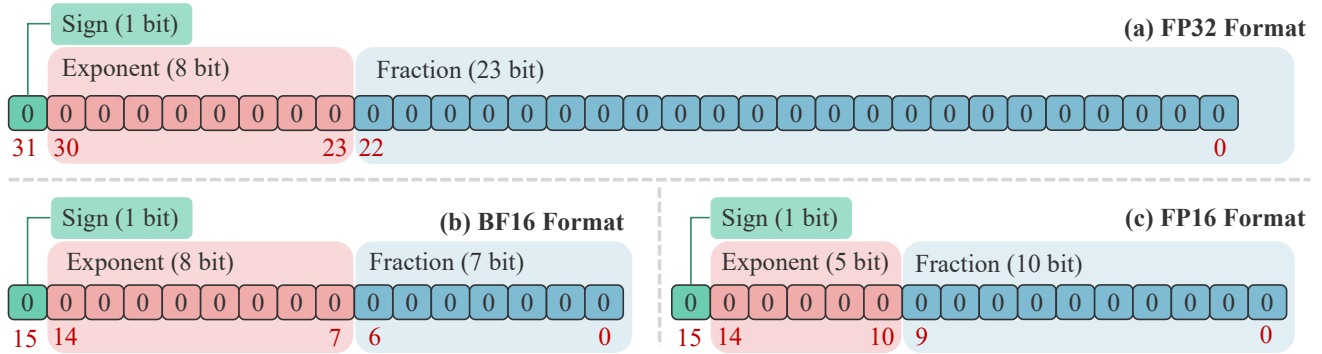


Figure 2: **Bit-level Composition and Comparison of Floating-Point Formats in Large Language Models.** (a) Standard Single-precision FP32, (b) Brain Floating-point BF16, and (c) Half-precision FP16. Specifically, the different color blocks denote the Sign (1-bit green), Exponent (red), and Fraction (blue) segments, respectively. While BF16 and FP16 are both 16-bit, BF16 features a wider exponent (8-bit) and narrower fraction (7-bit) compared to FP16 (5-bit exponent, 10-bit fraction).

Table 1: **Comprehensive Comparison of Four Representative Paradigms for LLM Lossless Compression.** We evaluate each method (see Table 4.1.2) across three key dimensions: compression efficiency (C-Ratio), computational latency (Time-Cost), and hardware requirements (Resource Usage). A greater number of “★” symbols signifies a more pronounced advantage in the corresponding metric.

Method	C-Ratio	Time Cost	Resource Usage	The Representative Works
Floating-Point-Number-Driven (FPMD)	★★	★★★	★★★	FPzip [39], MPC [40], SPDP [41], BUFF [42], Ndzip-C/GPU [43], PFPC [44]
Large-Language-Models-Driven (LLMD)	★★★★	★★	★★	ZipNN [22], FM-Delta [23], DFloat11 [45]
Learning-based Lossless Compression (LLC)	★★★★	★	★	MSDZip [26], MSDLC [46], DZip [25], TRACE [47], LLMZip [48], CMIX [49]
Traditional Lossless Compression (TLC)	★	★★★★	★★★★	ZPAQ [50], LZMA [51], ZSTD [52], BSC [53], PPMD [54; 55; 56], PIGZ [57]

2 Related Work

As shown in Fig 2, the LLMs are typically stored in float-point files using FP32, BF16, or FP16 formats. It consists of sign, exponent, and fraction [22]. Many existing reviews and benchmark studies have been proposed for compressing these float-point data, covering 1) lossy compression methods, 2) specific domain compressors, and 3) multi-model data compressors. Here, we briefly revisit those works and elucidate the distinctions between ours and them.

2.1 Lossy Compression Methods for LLMs

There are threefold methods, as our focus lies on lossless compressors that ensure data integrity, we provide a brief introduction to them here. 1) Pruning, it refers to the process of systematically removing redundant or non-critical components (such as weights, neurons, or layers) from LLMs [13]. The review and benchmark [58; 59; 60; 61] explore the impact of different pruning strategies on network performance. 2) Distillation, which involves transferring the knowledge from a complex Teacher Model to a smaller Student Model, facilitates model light-weighting and enhances efficiency. The [62; 63; 64] provide the latest reviews and benchmarks in this field. 3) Quantization, which optimizes inference and storage by reducing the numerical precision of parameters, includes methods such as [65; 66; 67; 68; 69; 70].

2.2 Specific Domains Compression Methods

These methods focus on specific data formats or application domains. FCBench [28] is the latest floating-point compression benchmark under high-performance computing platforms, which explores the overall performance of 8 CPU/GPU compressors across 33 datasets. Son et al. [29] explored the application of 12 compressors in scientific simulation check-pointing, tailored for the big-data-era. Wang et al. [30] conducted a thorough investigation into 21 compression approaches for database applications, utilizing both synthetic and real-world datasets. Cappello et al. [31] introduced innovations in floating-point dataset compression techniques, investigating their applicability in advancing scientific computing practices. Moreover, specialized compression studies in certain research domains,

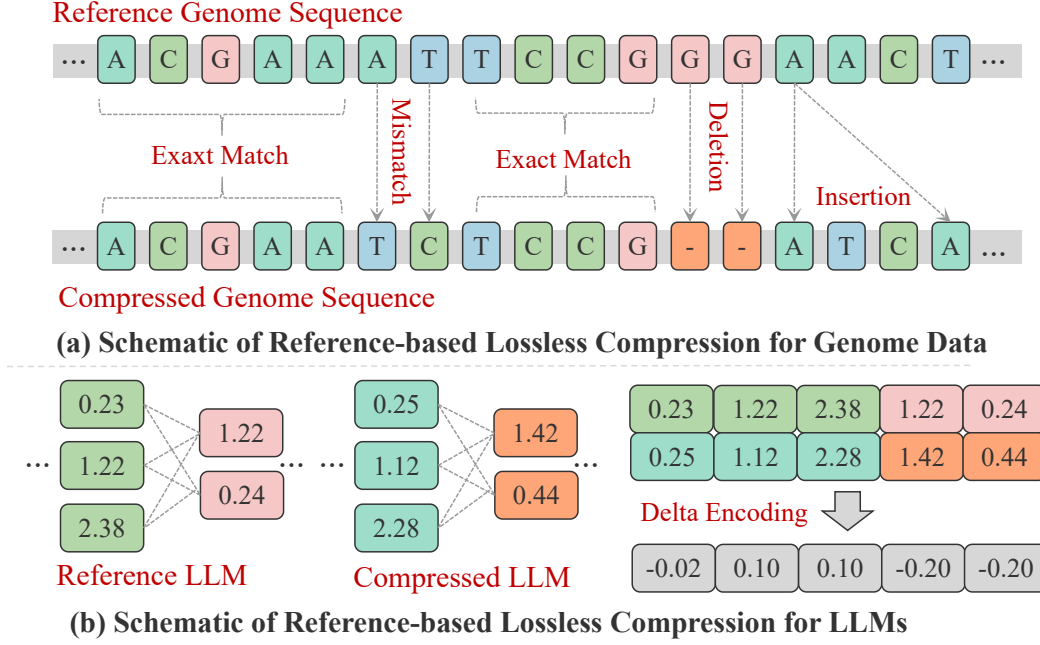


Figure 3: **Schematic diagram of reference-based compression. (a) Genome data, (b) Large language model parameters.** In sub-fig (a), reference-based mapping encompasses four modes: *exact match*, *mismatch*, *deletion*, and *insertion*. For exact matches “ACGAA” and “TCCG”, the method records the mode, start position, and length, i.e., $ACGAA \Rightarrow [0, 0, 5]$ and $TCCG \Rightarrow [0, 7, 4]$. Mismatched characters “A” and “T” are encoded with their specific positions: $A \Rightarrow [1, 5, T]$ and $T \Rightarrow [1, 6, C]$. The deletion “GG” is stored by its position and length: $GG \Rightarrow [2, 11, 2]$. Finally, the insertion at index 13 is recorded with its position and sequence: $A \Rightarrow [3, 13, ATCA]$. In sub-fig (b), reference-based LLM parameter compression records the differences between the two via delta encoding, which reduces the information entropy.

such as genomic data [36; 37; 71; 72; 73], medical data [74; 75; 76; 77], power grid data [78; 79; 80], and astronomical data [81; 82; 83], also provide valuable insights for the lossless compression of LLMs.

2.3 Multi-Model Data Compression Methods

These methods are universally applicable to lossless compression of multi-model data, including floating-point numbers, images, videos, text, et al. NNLCB [27] is the latest learning-based universal compression benchmark, which has comprehensively evaluated 17 lossless algorithms across 28 real-world datasets. SCB [84] systematically tests various lossless methods across 28 datasets, primarily focusing on traditional, non-learning-based approaches. LTCB [85] primarily focuses on the text domain and evaluates compression techniques across an impressive scale of 211 configurations. TurboBench [86] is a high-performance compression benchmark tool that supports more than 70 popular compressors. Kaur et al. [32] provided a systematic overview of the use of data deduplication techniques in lossless compression solutions tailored for cloud environments. Chiarot et al. [34] carried out an in-depth examination of time-series compressors, analyzing their classifications, assessment criteria, and real-world use cases. Jayasankar et al. [33] presented an extensive analysis of the evolution and practical use of compression techniques.

This paper introduces LLCBench as a valuable addition to the existing reviews and benchmark efforts. It focuses on the lossless compression of LLMs, conducting comprehensive evaluations of various compression tools, both specialized and general-purpose.

3 Lossless Compression Technologies for LLMs

Based on the specialized or general-purpose characteristics of compressors for LLMs compression, we categorize them into four types: the specialized 1) Floating-Point-Number-Driven (FPMD) and 2) Large-Language-Models-

Driven (LLMD) compressors, as well as the general-purpose 3) Learning-based Lossless Compression (LLC) and 4) Traditional Lossless Compression (TLC). Table 1 systematically compares the differences between these methods. This qualitative classification is derived from our experimental testing; for quantitative results, see Table 4.1.2 and B Table 30.

3.1 Floating-Point-Number-Driven (FPMD)

FPMD is a specialized algorithm designed for floating-point numbers, particularly useful for LLMs’ parameters. It is categorized into Non-prediction-based and Prediction-based methods.

3.1.1 Prediction-based Methods

This category encompasses methods based on predictive statistics and arithmetic coding [87; 88]. Lindstrom et al. [39] propose FPzip, a Lorenzo-predictor-based [89] compressor offering both lossless and lossy compression for floating-point data commonly used in scientific simulations. PFPC [44] is a parallel, prediction-based algorithm for double-precision scientific simulation data, leveraging hash tables for value prediction, XOR operations for residual calculation, and efficient encoding of leading zeros and nonzero residuals. Ndzip-CPU [90] is a prediction-based lossless compression algorithm optimized for high throughput. Its CPU version leverages SIMD and thread-level parallelism, while Ndzip-GPU [43] enhances parallelism by distributing transformation and residual coding across up to 768 GPU threads.

3.1.2 None-prediction-based Methods

This category primarily utilizes a combination of techniques such as dictionary coding [91], run-length encoding [92; 93], and Huffman coding [94] to achieve lossless compression. Liu et al. [42] introduce delta-based BUFF, which is specifically optimized for low-precision floating-point data and is widely utilized in server monitoring and Internet of Things (IoT) applications. SPDP [41] is a dictionary-based compressor that serves as both an HDF5 filter [95] and an independent compressor. It utilizes three transformation components to uncover data correlations and a reducer component to optimize encoding efficiency. Delta-based Gorilla [96] was proposed to efficiently compress timestamps and data points in Facebook’s in-memory time series database. Massive Parallel Compression (MPC) [40] is a delta-based algorithm optimized for floating-point data. By harnessing parallel processing, it significantly improves compression efficiency, making it ideal for high-performance computing tasks.

Typically, Prediction-based methods achieve higher compression ratios through probabilistic modeling but have lower throughput. In contrast, Non-reference-based methods prioritize throughput and resource efficiency at the cost of compression ratio.

3.2 Large-Language-Models-Driven (LLMD)

LLMD is a specialized method that leverages the characteristics of large language model parameters. These emerging algorithms include ZipNN [22], FM-Delta [23], DFloat11 [45], et al. Among these algorithms, we classify them into Reference-based and Non-reference-based approaches via biological data compression-based classification [71; 35; 37; 97; 98; 99], as shown in Fig. 3(a).

3.2.1 Reference-based Methods

It reduces redundancy by leveraging similarities between the “**reference**” and the to-be-compressed “**target**” LLMs, as shown in Fig. 3(b). Specifically, they use LLM $\mathcal{M}_{ref}$ as additional information and compare the differences between $\mathcal{M}_{tar}$ and $\mathcal{M}_{ref}$. These methods are similar to reference-based approaches in genomic data compression (see Fig. 3), with the difference being that LLMD utilizes a foundational LLM instead of a biological reference genome. These methods are inspired by two observations: 1) Fine-tuned LLMs in model repositories are derived from foundational model, making them particularly suitable for reference-based designs. 2) During model training, check-pointed versions exhibit minimal parameter updates, resulting in high similarity between successive snapshots. Ning et al. [23] propose FM-Delta, a specialized method designed for fine-tuned LLMs. Their analysis indicates that the similarity between the base LLM and its fine-tuned versions reaches 98%. Consequently, their algorithm consists of two steps: 1) Mapping floating-point numbers to integers to compute the difference between $\mathcal{M}_{ref}$ and $\mathcal{M}_{tar}$. 2) Compressing the difference using interval coding to generate a compact representation.

Experimental results show that FM-Delta reduces the storage requirement for the GPT-NeoX-20B [100; 101] family from 423 GB to 205 GB.

3.2.2 Non-reference-based Methods

These methods compress LLMs from scratch without requiring additional knowledge. Moshik et al. [22] propose ZipNN, which initially employs the Exponent Extraction Strategy (EES) - a floating-point grouping technique designed to refine the probability distribution of the disordered raw data. The grouped data is subsequently processed using ZSTD [52] or Huffman coding [94] to generate a compact representation. Experimental results demonstrate that ZipNN achieves up to a 17% improvement in compression ratio for Llama-3 [102] compared to general-purpose compressors. Zhang et al. [45] introduce DFloat11, a GPU-accelerated compression scheme leveraging entropy coding [94]. By assigning dynamic-length encoding to weights based on frequency, DFloat11 optimizes compression while preserving precision. To facilitate efficient inference, the authors developed a custom GPU kernel for real-time decompression. Experimental results indicate that DFloat11 reduces data storage up to 30%. Patrick et al. [103] introduce Huff-LLM, a comprehensive lossless compression method for LLM weights, enabling storage in a compressed format across diverse environments, including cloud, disk, main memory, and on-chip memory/buffers. Hao et al. [104] present NeuZip, a weight compression approach leveraging the entropy of floating-point numbers in LLMs. It enables memory-efficient training and inference while maintaining performance. Notably, the NeuZip reduces the memory of Llama-3 [102] training from 31 GB to 16 GB.

Reference-based methods achieve higher compression ratios for highly similar models; however, they introduce additional knowledge, limiting their widespread adoption. Non-reference-based methods, in contrast, leverage the inherent redundancy in LLM, offering a balanced trade-off between compression ratio and time cost.

3.3 Learning-based Lossless Compressors (LLC)

LLC belongs to context-based approach that utilizes neural networks to predict the distribution of the inputted data [27]. These predictions are then passed to arithmetic encoder [105; 87; 88; 106] to generate compact representation. LLC include three types: Static, Dynamic, and Semi-adaptive [107; 108; 27; 109].

3.3.1 Static Compression Methods

It consists of two steps: 1) Pre-training a static model $\mathcal{M}_{sta}$ on the inputted data; 2) Utilizing $\mathcal{M}_{sta}$ to compress the inputted LLM and saving the model as part of the compressed file. The distinction among most Static LLCs lies in the design of their architecture of $\mathcal{M}_{sta}$. DeepZip [110] is one of the earlier Static framework, utilizing Bidirectional Gated Recurrent Unit (BiGRU) [111] and Recurrent Neural Network (RNN) [112]. The subsequent DecMac [113] implementation is based on Long Short-Term Memory (LSTM) [114], which is a type of Recurrent Neural Network (RNN) designed to handle sequences with long-term dependencies effectively. Recently, researchers have developed compressors based on LLMs, including LLaMA-based [102] LLMZip [48], as well as LMIC [115], which incorporates the Chinchilla [116] LLM and Transformers [117] architecture.

3.3.2 Dynamic Compression Methods

This methods dynamically update the parameters of model $\mathcal{M}_{dyn}$ during the data compression process without requiring any pre-training. CMIX [49] is a dynamic compressor known for its exceptional compression ratio. It achieves this by blending over 2000 probabilistic prediction models. However, this impressive performance comes at the cost of significantly reduced compression throughput. Subsequently, the authors propose Lstm-compress [118] based on LSTM. Bellar et al. propose NNCP [119], which is based on LSTM and Transformer architectures. Subsequently, Trace [47] uses a similar architectural design. In later studies, Mao et al. develop OREO [120], a neural-network-based compressor featuring Multi-Layer Perceptron (MLP) [121] and Ordered Masks (OM). Building on the OREO algorithm, the authors later introduce PAC [122], a compression method designed to address issues of redundancy and intra-batch distribution variation. Ma et al. propose xLSTM-based [123] MSDLC [46] and MLP-based MSDZip [26]. As the latest state-of-the-art dynamic LLCs, these approaches also integrate Parallel Expansion Mapping (PEM) and Stepwise-parallel Mechanisms (SpM) to accelerate computational processes. FADE [124] is a cutting-edge deep learning-based lossless compressor that balances modeling precision with system efficiency. Through a novel dual-stream architecture, it decouples data into micro-syntactic and macro-semantic features. To eliminate the serial bottleneck of auto-regressive decoding, FADE employs a concurrent stream-parallel pipeline, achieving a significant breakthrough in throughput.

Table 2: **Detailed Information of All Used Large Language Models Datasets in Our Proposed LLCBench.**

ID	Model Name	Model Size (B)	Short Description
D_0	Qwen3-8B	16,381,517,176	A fine-tuned general-purpose text generation model
D_1	DeepSeek-R1-Distill-Qwen-7B	15,231,272,120	A distilled model optimized for efficient reasoning and logic tasks
D_2	Gpt2-medium-pubmed	1,418,253,632	A finetuned biomedical text generation model
D_3	Bert-large-finetuned-ner	1,337,252,416	One finetuned token classification (NER) model
D_4	Taiwan-ELM-270M-Instruct	1,236,873,728	A collection of Taiwanese instruction model
D_5	Qwen2.5-0.5B-Instruct	987,748,864	One causal language model
D_6	Git-base-fashion	706,479,488	One fashion vision-language model
D_7	SmolLM-135M-de	538,990,848	A finetuned language model for German
D_8	Comment-Moderation	267,641,344	One comment moderation model
D_9	Dinov2-small-finetuned	88,306,432	A well finetuned image classification model

3.3.3 Semi-adaptive Compression Methods

Such methods remain underexplored due to the complexity of model architecture design [27]. DZip (Supporter Mode) [25] stands as the sole advocate for these approaches. Specifically, semi-adaptive methods combine both static and dynamic approaches, consisting of three steps: 1) Pre-training a small-scale static model $\mathcal{M}_{sta}$ on the LLMs to be compressed; 2) Randomly initializing a dynamic model $\mathcal{M}_{dyn}$; 3) Freezing $\mathcal{M}_{sta}$ and updating $\mathcal{M}_{dyn}$ during compression process, utilizing their combined probabilities for data compression.

Static LLCs offer superior compression ratios but lower throughput, while Dynamic LLCs deliver higher throughput at the expense of compression ratios. Semi-adaptive LLCs aim to strike a balance between the two; however, their architectural design presents the greatest challenge.

3.4 Traditional Lossless Compression (TLC)

The TCL is optimized evolutions of fundamental compression technologies such as LZ77 [91], LZ78 [125], and BWT [126; 127], along with entropy coding algorithms like Huffman coding [94], arithmetic coding [87; 88], and asymmetric numeral systems [128]. Basic compression techniques minimize redundancy by identifying repeated patterns. For instance, LZ77 detects and replaces strings to reduce storage costs. Entropy coding algorithms [94; 87] further enhance compression efficiency by dynamically assigning code lengths based on symbol or pattern frequencies. TCLs can be classified into three types: Dictionary-based, Transfer-based, and Statistical-based [35; 129].

3.4.1 Dictionary-based Methods

LZMA [51], Brotli [130], LZ4 [131], GZip [132], ZSTD [52], Brieflz [133], Lizard [134], and LZO [135] are general-purpose compression algorithms that build upon the LZ77 framework. Brotli, GZip, Lizard, and ZSTD incorporate Huffman Coding alongside LZ77 to achieve enhanced compression efficiency. PigZ [57] is a parallelized variant of GZip, utilizes the same DEFLATE [136] (LZ77 + Huffman coding) but accelerates processing by dividing large files into multiple blocks via multi-threading. LZOP [137], derived from LZO, employs LZ77 with rapid dictionary lookups for compression, refining algorithmic details to optimize speed and dictionary management. LZMA replaces Huffman coding with range coding [138], offering superior compression ratios at the expense of reduced speed, while LZMA2 [51; 139] extends LZMA with multi-core processing support for parallelized compression. SnZip [140], based on the Snappy [141] algorithm, which is designed for ultra-fast decompression by minimizing computational overhead.

3.4.2 Transfer-based Methods

These methods enhance compression efficiency by rearranging raw data and integrating existing encoding techniques. For example, BZip2 [142] and its variant BSC [53] employ Burrows-Wheeler Transform (BWT) [126; 127] and Move-To-Front (MTF) [143] algorithms with Huffman coding. Here, BWT reorders the original LLM for re-building data entropy in the local area, while MTF and Huffman coding further reduce redundancy for efficient compression. PBZip2 [144] is a parallelized implementation of BZip2, retains BWT and MTF-based compression but significantly accelerates processing by splitting data into multiple segments using multi-threading [129].

Table 3: **Details of all Tested Algorithms in LLCBench.** “FPMD”, “LLMD”, “LLC”, and “TLC” denote Floating-Point-Number-Driven, Large-Language- Models-Driven, Learning-based Lossless Compression, Traditional Lossless Compression method, respectively. “Year” refers to the publication year or the latest version used. “Hyper-parameters” is the parameters used in the algorithm’s execution. These values are determined via their default settings. “CPU/GPU” indicates the computing hardware used by the algorithm.

Compression Algorithm	Year	Type	Version	Language	CPU / GPU?	Hyper-parameters
MPC [40]	2015	FPMD	1.3.0	C / CUDA	CPU / GPU	-1
FPZip [39]	2019	FPMD	1.3.0	C / C++	CPU	
SPDP [41]	2018	FPMD	1.1	C / C++	CPU	-10
FM-Delta [23]	2025	LLMD	1.0.0	Python / C++	CPU	
ZipNN [22]	2025	LLMD	0.2.0	Python / C++	CPU	
LZMA [51]	2023	TLC	23.01	C / C++	CPU	-m0=lzma -mx9 -mmt16
LZMA2 [139]	2023	TLC	23.01	C / C++	CPU	-m0=lzma2 -mx9 -mmt16
PPMD [56]	2023	TLC	23.01	C / C++	CPU	-m0=ppmd -mx9 -mmt16
LZOP [137]	2017	TLC	1.04	C / C++	CPU	-9
BZip2 [145]	2024	TLC	1.0.8	C / C++	CPU	-kzfc9
PBZip2 [144]	2016	TLC	1.1.6	C / C++	CPU	-9c -m2000 -p16
Lizard [134]	2017	TLC	1.0.0	C / C++	CPU	-49
BSC [53]	2016	TLC	3.4.4	C / C++	CPU	-e2
Brotli [130]	2018	TLC	1.1.0	C / C++	CPU	
LZO [135]	2017	TLC	2.10	C / C++	CPU	-9
PIGZ [57]	2013	TLC	2.8	C / C++	CPU	-cf -9
SnZip [140]	2021	TLC	1.0.5	C / C++	CPU	-k -t
ZSTD [52]	2021	TLC	1.5.6	C / C++	CPU	-19
ZPAQ [50]	2009	TLC	7.15	C / C++	CPU	-method 5
MSDZip [26]	2025	LLC	1.0.1	Python	CPU + GPU	-bs 512 -ts 16 -p 1
DeepZip [110]	2019	LLC	1.0.0	Python	CPU + GPU	-bs 8192 -ts 32
DZip [25]	2021	LLC	1.0.0	Python	CPU + GPU	-bs 8192 -ts 32
PAC [122]	2022	LLC	1.0.0	Python	CPU + GPU	-bs 512 -ts 32
TRACE [47]	2022	LLC	1.0.0	Python	CPU + GPU	-bs 512 -ts 32

3.4.3 Statistical-based Methods

These methods are generally used in conjunction with arithmetic coding [87; 88], and some of the previously mentioned LLCs also fall into this subcategory. Representative statistical-based TCL algorithms include ZPAQ [50] and PPMD [54; 55; 56]. PPMD is a statistical solution based on Prediction by Partial Matching (PPM) [54]. It enhances text encoding by building a predictive model that estimates the probability of each character using prior occurrence patterns. ZPAQ [50], on the other hand, employs an incremental compression strategy leveraging dictionary and block reuse. It specializes in incremental backups by compressing only newly added or modified data, avoiding unnecessary re-compression of unchanged content.

4 Benchmark Testing and Analysis

This section first presents the experimental configurations of LLCBench, and then delves into the assessment of compression performance, runtime efficiency, and resource consumption.

4.1 LLCBench Setups

All benchmark experiments were performed on a same Ubuntu 20.04.6 LTS server featuring 2 Intel Xeon Gold 6248 CPUs (2.50 GHz, 40 cores combined), 6 NVIDIA GeForce RTX 4090 GPUs (16,384 CUDA cores and 24 GB memory each), and 314 GB of DDR4 RAM.

4.1.1 Experimental Datasets

As illustrated in Table 2, we curated the initial version of LLCBD (**LLCBench Datasets**) by selecting 10 representative groups of Large Language Models from the Hugging Face Model Hub⁵, including Qwen⁶, DeepSeek⁷,

⁵<https://huggingface.co/models>

⁶<https://huggingface.co/Qwen/Qwen3-8B> and <https://huggingface.co/Gensyn/Qwen2.5-0.5B-Instruct>

⁷<https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B>

Gpt ⁸, Bert ⁹, Taiwan-ELM ¹⁰, Git ¹¹, SmolLM ¹², Comment-Moderation ¹³, and Dinov2 ¹⁴. These models were specifically chosen for their substantial download volumes and widespread community adoption, ensuring the benchmark’s practical relevance. The proposed LLCBD encompasses a diverse range of prominent architectures, spanning multiple heterogeneous modalities such as text, video, speech, and imagery.

4.1.2 Tested Algorithms

As shown in Table 3, in our LLCBench, we tested 24 state-of-the-art lossless methods for LLMs compression, including FPMD-based MPC (FCBench Version ¹⁵), FPZip ¹⁶, and SPDP ¹⁷; LLMD-based FM-Delta ¹⁸ and ZipNN ¹⁹; LLC-based MSDZip ²⁰, DeepZip ²¹, DZip ²², PAC ²³ and TRACE ²⁴; as well as TLC-based LZMA [51], LZMA2 ²⁵, PPM [54], LZOP ²⁶, BZip2 ²⁷, PBZip2 ²⁸, Lizard ²⁹, BSC ³⁰, Brotli ³¹, LZO ³², PIGZ ³³, SnZip ³⁴, ZSTD ³⁵, and ZPAQ ³⁶. A.1 to A.24 shows the details of the all tested lossless compression algorithms for LLMs in our proposed LLCBench.

4.1.3 Evaluation Metrics

We tested the overall performance of the 24 compression methods across three dimensions: lossless compression performance, throughput, and resource consumption [27; 39; 146; 147; 8; 35]. The first category consists of five compression performance indicators, including: the Average and Weighted Average Compression Ratio (Avg/WAvgCR $\uparrow$):

$$Avg/WAvgCR = \sum_{i=0}^{n-1} \alpha_i \times \frac{F_i^b}{F_i^a} \quad (1)$$

the Average and Weighted Average Storage Saving Percentage (Avg/WavgSSP $\uparrow$):

$$Avg/WavgSSP = \sum_{i=0}^{n-1} \alpha_i \times \left(\left(1 - \frac{F_i^a}{F_i^b} \right) \times 100\% \right) \quad (2)$$

and the Compression Robustness Performance (CRP $\downarrow$):

$$CRP = \frac{\sqrt{\sum_{i=0}^{n-1} (CR_i - CR_u)^2}}{\sqrt{n} \times CR_u} \times 100\% \quad (3)$$

⁸<https://huggingface.co/cwestnedge/gpt2-medium-pubmed>

⁹<https://huggingface.co/Jorgeutd/bert-large-uncased-finetuned-ner>

¹⁰<https://huggingface.co/liswei/Taiwan-ELM-270M-Instruct>

¹¹<https://huggingface.co/swaroopajit/git-base-fashion>

¹²<https://huggingface.co/LemiSt/SmolLM-135M-de>

¹³<https://huggingface.co/Vrandan/Comment-Moderation>

¹⁴<https://huggingface.co/matthieulel/dinov2-small-finetuned-galaxy10-decals>

¹⁵<https://github.com/hdpds-group/FCBench>

¹⁶<https://github.com/llnl/fpzip>

¹⁷<https://userweb.cs.txstate.edu/~burtcher/research/SPDPcompressor>

¹⁸<https://github.com/ningwanyu/FM-Delta>

¹⁹<https://github.com/zipnn/zipnn>

²⁰<https://github.com/mhuidong/MSDZip>

²¹<https://github.com/mohit1997/DeepZip>

²²<https://github.com/mohit1997/Dzip-torch>

²³https://github.com/mynotwo/compressor_via_simple_and_scalable_parameterization

²⁴<https://github.com/mynotwo/A-Fast-Transformer-based-General-Purpose-LosslessCompressor>

²⁵<https://www.7-zip.org/sdk.html>

²⁶<https://www.lzop.org>

²⁷<https://sourceware.org/bzip2>

²⁸<https://linux.die.net/man/1/pbzip2>

²⁹<https://github.com/inikep/lizard>

³⁰<https://github.com/IlyaGrebnov/libbsc>

³¹<https://github.com/google/brotli>

³²<https://www.oberhumer.com/opensource/lzo>

³³<https://linux.die.net/man/1/pigz>

³⁴<https://github.com/kubo/snzip>

³⁵<https://github.com/facebook/zstd>

³⁶<https://mattmahoney.net/dc/zpaq.html>

Table 4: Overall performance of tested algorithms on benchmark datasets. Detailed per-algorithm results appear in Appendix A (Tables 6–29). Green, pink, and light blue mark the top three results. AvgCR and WAvGCR denote mean and weighted mean compression ratios; AvgSSP and WAvGSSP denote mean and weighted mean storage-saving percentages; CRP denotes compression robustness; TotalCT and TotalDT denote compression and decompression time in hours; AvgCPM and AvgDPM denote average peak CPU memory usage for compression and decompression.

Algorithm / Metric	AvgCR (Ratio)	WavgCR (Ratio)	AvgSSP (%)	WAvGSSP (%)	CRP (%)	TotalCT (Hours)	TotalDT (Hours)	AvgCPM (GB)	AvgDPM (GB)
FM-Delta [23]	2.038	1.571	32.789	27.240	88.936	0.031	0.032	0.690	0.846
MSDZip [26]	1.494	1.420	26.466	24.630	40.681	115.142	114.258	13.413	14.215
PAC [122]	1.494	1.420	26.448	24.615	40.689	57.104	71.617	13.405	14.110
ZPAQ [50]	1.489	1.415	26.185	24.362	40.694	0.808	0.819	8.286	7.697
TRACE [47]	1.488	1.414	26.173	24.336	40.579	46.786	70.071	13.405	14.110
DZip [25]	1.449	1.395	24.346	23.688	39.638	232.644	81.540	15.008	8.848
DeepZip [110]	1.449	1.395	24.342	23.685	39.637	169.898	23.369	51.028	8.484
BSC [53]	1.434	1.361	22.711	20.862	42.677	0.029	0.020	4.479	2.561
PPMD [56]	1.378	1.307	19.305	17.329	43.456	1.051	1.118	0.257	0.256
LZMA [51]	1.373	1.306	19.827	17.952	40.969	0.671	0.061	0.667	0.069
LZMA2 [139]	1.368	1.301	19.648	17.735	40.566	0.233	0.021	3.264	1.412
Brotli [130]	1.346	1.281	18.555	16.572	39.901	4.943	0.021	0.215	0.019
ZipNN [22]	1.314	1.276	19.901	18.528	27.155	0.008	0.006	3.164	1.799
BZip2 [145]	1.343	1.272	16.650	14.586	45.006	0.193	0.102	0.007	0.005
PBZip2 [144]	1.343	1.272	16.644	14.580	45.001	0.015	0.010	0.133	0.110
MPC [40]	1.292	1.244	17.933	16.205	30.803	0.004	0.004	1.506	1.506
ZSTD [52]	1.302	1.243	16.907	14.914	36.518	0.641	0.004	0.256	0.012
PIGZ [57]	1.256	1.208	15.539	13.662	30.702	0.017	0.010	0.014	0.002
FPZip [39]	1.124	1.130	10.761	11.125	6.163	0.040	0.036	0.897	0.879
Lizard [134]	1.176	1.118	8.025	5.522	37.262	0.368	0.004	0.170	0.040
LZOP [137]	1.136	1.095	7.513	5.487	27.904	0.237	0.004	0.002	0.002
LZO [135]	1.135	1.094	7.488	5.472	27.696	0.219	0.004	0.002	0.002
SPDP [41]	1.120	1.078	6.139	3.987	28.283	0.010	0.011	0.018	0.018
SnZip [140]	1.055	1.036	3.851	2.524	13.798	0.005	0.003	0.003	0.003

Table 5: **Average Peak GPU Memory usage for GPU-Accelerated Compression Methods.** The instance # $\{1, 2, 3\}$ denote the top-3 compression performance in the GPU memory usage.

Metric / GPU-Accelerated Algorithm	DeepZip [110]	Dzip [25]	PAC [122]	TRACE [47]	MSDZip [26]	MPC [122]
Avg-GPM (MB, Megabyte)	474.758 ^{#3}	2673.051	395.428 ^{#2}	795.849	1896.001	1.942 ^{#1}

Among the Eq. (1) and Eq. (2), the n denote the total number of tested datasets for LLMs, the F_i^b and F_i^a denote the i -th file size of LLM before and after lossless compression, α_i represent the weighted factor for the i -th file. For the AvgCR, $\alpha_i = \frac{1}{n}$, for the WAvGCR, $\alpha_i = \frac{F_i^b}{\sum_{j=0}^{n-1} F_j^b}$. In Eq. (3), $CR_i = \frac{F_i^b}{F_i^a}$ and $CR_u = \frac{CR_i}{n}$ denote the i -th and average compression ratio value, respectively.

The second category, two throughput performance indicators, measures the overall time taken for compression and decompression (TotalCT/DT $\downarrow$). The third category, three resource consumption indicators, evaluates average CPU memory usage during both compression and decompression (AvgCPM/DPM $\downarrow$), as well as peak average GPU memory consumption (AvgGPM $\downarrow$).

4.2 Compression Performance

As shown in Table 4.1.2, the top-3 methods in overall compression ratio performance of AvgCR and WAvGCR are LLMD-based FM-Delta (2.038 & 1.571) [23], LLM-based MSDZip (1.494 & 1.420) [26] and PAC (1.494 & 1.420) [122], as well as TLC-based ZPAQ [50] (1.489 & 1.415), respectively ³⁷. Compared to the weakest performer, SnZip [140], the improvement ($\frac{CR(\text{baseline}) - CR(\text{SnZip})}{CR(\text{SnZip})} \times 100\%$) in AvgCR are 93.175%, 41.611%, 41.611%, and

³⁷For a fair comparison, the results in Table 4.1.2 are aggregated based on the $D_2 \sim D_9$, as several methods failed to execute successfully on the larger-scale D_0 and D_1 . It should be noted that excluding D_0 and D_1 for all evaluated methods does not alter the core conclusions of this study. For a more exhaustive comparison across the entire dataset collection ($D_0 \sim D_9$), see B Table 30.

41.137% , while the enhancement in WAvgCR are 51.641%, 37.066%, 37.06% and 36.58%, respectively. The top-3 methods reduce AvgSSP by 32.789%, 26.466%, and 26.448%, while they lower WAvgSSP by 27.240%, 24.630%, and 24.615%, respectively. Based on these results, we reach three conclusions: 1) The reference-based LLMD approach (FM-Delta) effectively reduces the scale of LLMs, indicating that pretrained and fine-tuned models have minimal parameter similarity differences. 2) LLC-based methods (MSDZip and PAC) demonstrate competitive advantages, indicating that deep network architectures still have potential for compressing LLM parameters. 3) Traditional context modeling methods (ZPAQ) remain a solid choice, as they offer a balanced trade-off between compression ratio and processing time.

Although FM-Delta, MSDZip, PAC, and ZPAQ excel in reducing the original scale, their compression robustness performance (CRP) is not the best. The top three methods in CRP are FPZip [39] (6.163%), SnZip [140] (13.798%), and ZipNN [22] (27.155%), respectively. This indicates that LLC-based and FPMD-based methods are more susceptible to the probability distribution of the dataset.

4.3 Running Time

As shown in Table 4.1.2, the top-3 algorithms in the overall compression time are MPC [40], SnZip [140], and ZipNN [22], while the top-3 in decompression time are SnZip, MPC [40], ZSTD [52], Lizard [134], LZOP [137], LZO [135], PBZip2 [144], and PIGZ [57], respectively. LLC-based methods excel in compression ratio but have a significant drawback in processing time. The total runtime for MSDZip [26], PAC [122], TRACE [47], DZip [25], and DeepZip [110] reaches 229.400, 128.721, 116.857, 314.184, and 193.267 hours, respectively. Unfortunately, the substantial runtime of LLC-based methods makes them impractical for industrial applications. However, they remain viable for long-term backup of large datasets. Additionally, with advancements in quantum computing and other emerging technologies [148], LLC-based methods may become a core component of future compression systems. Additionally, we find that the reference-based LLMD method, FM-Delta, performs remarkably well (0.031 hours for compression and 0.032 hours for decompression), primarily due to its core floating-point encoding component, FPZip [39]. However, FM-Delta introduces additional reference information, which results in reduced flexibility, especially when compressing non-fine-tuned models.

4.4 Memory Resource Usage

Table 4.1.2 presents the average peak CPU memory usage for compression and decompression across 24 algorithms. LZO (0.002 GB), LZOP (0.002 GB), SnZip (0.003 GB), and BZip2 (0.007 GB) rank in the top-3 for AvgCPM, while LZO (0.002 GB), LZOP (0.002 GB), PIGZ (0.002 GB), SnZip (0.003 GB), and BZip2 (0.005) are the top-3 performers in AvgDPM. Considering that some algorithms utilize GPUs, such as TRACE, PAC, and MSDZip, Table 5 also provides their CUDA memory consumption. Among them, the top-3 are MPC (1.942 MB), PAC (395.425 MB), and DeepZip (474.758 MB).

This suggests that traditional methods have the potential for robust performance on devices with extremely limited computational resources, such as smartwatches, rings, and brain-machine interfaces. Additionally, context modeling-based methods (such as LLC-based MSDZip, PAC, TRACE, DeepZip, and DZip, as well as TLC-based ZPAQ and LZMA2 [139]) also consume substantial resources. However, for large-memory servers, sacrificing relatively inexpensive system resources in exchange for higher compression efficiency may be worthwhile.

5 Future Direction for LLMs Lossless compression

Based on our review, benchmark results, as well as existing literature [27; 35], we summarize the following potential research directions: 1) Automated Parameter Tuning, 2) Smart Storage and Distribution; 3) Compression Parallel Acceleration, and 4) Robust Algorithm Design.

5.1 Automated Parameter Tuning

Large-language-model-driven and learning-based lossless compressors increasingly prioritize multi-objective optimization to navigate the complex design space of neural compression. Representative frameworks such as TRACE [47] and MSDZip [26] have pioneered this approach by integrating multi-parameter configurations that simultaneously account for bit-rate reduction (compression ratio) and computational overhead (resource consumption). However, the interplay between these dimensions is often non-linear. By implementing sophisticated optimization strategies, such as dynamic resource allocation or adaptive parameter pruning, it is possible to strike

a finer balance across the performance spectrum (Examples include the automated adjustment of context length for LLMs, GPU batch size, and model learning rates.). Such advancements not only refine compression efficiency but also substantially reduce processing latency, making LLMDs and LLCs lossless compressors more viable for real-world high-throughput scenarios.

5.2 Smart Storage and Distribution

The storage and sharing of LLMs should be considered holistically within the model storage system. Leveraging advanced LLM-based agents [149; 150; 7] to automatically select the most suitable compressor based on user and resource requirements remains an area for further research. For examples, as illustrated in Table 4.1.2, the future compression Agent system can adaptively select FM-Delta [23] (achieving a compression ratio of 2.038) when the reference model is accessible. Alternatively, it may opt for MSDZip [26] in scenarios where time constraints are negligible, or prioritize MPC [40], despite its lower compression ratio, due to its minimal time and memory footprint.

5.3 Compression Parallel Acceleration

Learning-based compression methods, such as compressors TRACE [47] (46.786 hours for compression and 70.071 hours for decompression) and DZip [25] (232.644 hours for compression and 81.540 hours for decompression), exhibit high overall execution times. Exploring multi-GPU computation or designing parallel compression algorithms optimized for new hardware architectures could significantly improve time efficiency while maintaining high compression ratios [151; 99; 105; 152]. Such advancements would enhance the practical usability of these compressors, particularly for large-scale applications.

5.4 Robust Algorithm Design

As shown in the in the Table 4.1.2, the best compression ratio algorithm, FM-Delta [23] (2.038), exhibits the weakest robustness (88.936%), indicating a high sensitivity to the probability distribution of the dataset (see Eq.(3)). Exploring the impact of data probability distribution on algorithm performance and designing adaptable compression structures to enhance robustness is a worthwhile research direction. A recent endeavor, LLM-Agent-driven AgentGC [8], introduces an evolutionary compression framework for genomic sequences by integrating online LLM Agents. It features a synergistic optimization of dataset probability distributions, system resources, and algorithmic parameters. This methodology can be extended to lossless LLM compression, particularly within the contexts of data transmission and collaborative sharing.

6 Limitations and Conclusion

In this paper, we proposed the first lossless compression review and benchmark specifically for large language models. Specifically, we first introduced a systematic classification method, dividing existing works into four categories (Floating-Point-Driven, Large-Language-Model-Driven, Learning-Based Lossless Compression, and Traditional Lossless Compression) and evaluating their advantages and disadvantages from multiple dimensions. Then, we presented a comprehensive benchmark, assessing 10 performance metrics across 24 compressors on 10 widely used LLM datasets. Finally, we identified future research directions for lossless compression of LLMs. Currently, the scale of algorithms and datasets in LLCBench can still be further improved. To address this limitation, we will collect more algorithms and datasets in the future and continue to maintain this benchmark.

References

- [1] Agrim Gupta, Silvio Savarese, Surya Ganguli, and Li Fei-Fei. Embodied intelligence via learning and evolution. *Nature communications*, 12(1):5721, 2021.
- [2] David Howard, Agoston E Eiben, Danielle Frances Kennedy, Jean-Baptiste Mouret, Philip Valencia, and Dave Winkler. Evolving embodied intelligence from materials to machines. *Nature Machine Intelligence*, 1(1):12–19, 2019.

- [3] Gianmarco Mengaldo, Federico Renda, Steven L Brunton, Moritz Bächer, Marcello Calisti, Christian Duriez, Gregory S Chirikjian, and Cecilia Laschi. A concise guide to modelling the physics of embodied intelligence in soft robotics. *Nature Reviews Physics*, 4(9):595–610, 2022.
- [4] Richard Van Noorden and Jeffrey M Perkel. Ai and science: what 1,600 researchers think. *Nature*, 621(7980):672–675, 2023.
- [5] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Židek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.
- [6] Yongjun Xu, Xin Liu, Xin Cao, Changping Huang, Enke Liu, Sen Qian, Xingchen Liu, Yanjun Wu, Fengliang Dong, Cheng-Wei Qiu, et al. Artificial intelligence: A powerful paradigm for scientific research. *The Innovation*, 2(4), 2021.
- [7] Hui Sun, Chang Xu, Haonan Xie, Hao Li, Yuhao Huang, Chuheng Zhang, Ming Jin, Xiaoguang Liu, Gang Wang, and Jiang Bian. Chatad: Reasoning-enhanced time-series anomaly detection with multi-turn instruction evolution. *arXiv preprint arXiv:2601.13546*, 2026.
- [8] Sun Hui, Ding Yanfeng, Huidong Ma, Chang Xu, Keyan Jin, Lizheng Zu, Cheng Zhong, Gang Wang, Wentong Cai, et al. Agentgc: Evolutionary learning-based lossless compression for genomics data with llm-driven multiple agent. *arXiv preprint arXiv:2601.13559*, 2026.
- [9] Satyadhar Joshi. Review of gen ai models for financial risk management. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 11(1):709–723, 2025.
- [10] Minghong Fan. Llms in banking: Applications, challenges, and approaches. In *Proceedings of the International Conference on Digital Economy, Blockchain and Artificial Intelligence*, pages 314–321, 2024.
- [11] Huaqin Zhao, Zhengliang Liu, Zihao Wu, Yiwei Li, Tianze Yang, Peng Shu, Shaochen Xu, Haixing Dai, Lin Zhao, Gengchen Mai, et al. Revolutionizing finance with llms: An overview of applications and insights. *arXiv preprint arXiv:2401.11641*, 2024.
- [12] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.
- [13] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.
- [14] Max Marion, Ahmet Üstün, Luiza Pozzobon, Alex Wang, Marzieh Fadaee, and Sara Hooker. When less is more: Investigating data pruning for pretraining llms at scale. *arXiv preprint arXiv:2309.04564*, 2023.
- [15] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100, 2024.
- [16] Wei Huang, Xingyu Zheng, Xudong Ma, Haotong Qin, Chengtao Lv, Hong Chen, Jie Luo, Xiaojuan Qi, Xianglong Liu, and Michele Magno. An empirical study of llama3 quantization: From llms to mllms. *Visual Intelligence*, 2(1):36, 2024.
- [17] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.
- [18] Lei Li, Yongfeng Zhang, and Li Chen. Prompt distillation for efficient llm-based recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 1348–1357, 2023.
- [19] Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*, 2024.

- [20] Chuanpeng Yang, Yao Zhu, Wang Lu, Yidong Wang, Qian Chen, Chenlong Gao, Bingjie Yan, and Yiqiang Chen. Survey on knowledge distillation for large language models: methods, evaluation, and application. *ACM Transactions on Intelligent Systems and Technology*, 2024.
- [21] Jiaheng Liu, Chenchen Zhang, Jinyang Guo, Yuanxing Zhang, Haoran Que, Ken Deng, Jie Liu, Ge Zhang, Yanan Wu, Congnan Liu, et al. Ddk: Distilling domain knowledge for efficient large language models. *Advances in Neural Information Processing Systems*, 37:98297–98319, 2024.
- [22] Moshik Hershcovitch, Andrew Wood, Leshem Choshen, Guy Girmonsky, Roy Leibovitz, Ilias Ennmouri, Michal Malka, Peter Chin, Swaminathan Sundararaman, and Danny Harnik. Zipnn: Lossless compression for ai models. *arXiv preprint arXiv:2411.05239*, 2024.
- [23] Wanyi Ning, Jingyu Wang, Qi Qi, Mengde Zhu, Haifeng Sun, Daixuan Cheng, Jianxin Liao, and Ce Zhang. Fm-delta: Lossless compression for storing massive fine-tuned foundation models. *Advances in Neural Information Processing Systems*, 37:66796–66825, 2024.
- [24] Nikhil Kandpal, Brian Lester, Mohammed Muqeeth, Anisha Mascarenhas, Monty Evans, Vishal Baskaran, Tenghao Huang, Haokun Liu, and Colin Raffel. Git-theta: A git extension for collaborative development of machine learning models. In *International Conference on Machine Learning*, pages 15708–15719. PMLR, 2023.
- [25] Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. Dzip: improved general-purpose lossless compression based on novel neural network modeling. In *2021 data compression conference (DCC)*, pages 153–162. IEEE, 2021.
- [26] Huidong Ma, Hui Sun, Liping Yi, Yanfeng Ding, Xiaoguang Liu, and Gang Wang. Msdzip: Universal lossless compression for multi-source data via stepwise-parallel and learning-based prediction. In *Proceedings of the ACM on Web Conference 2025*, pages 3543–3551, 2025.
- [27] Hui Sun, Huidong Ma, Feng Ling, Haonan Xie, Yongxia Sun, Liping Yi, Meng Yan, Cheng Zhong, Xiaoguang Liu, and Gang Wang. A survey and benchmark evaluation for neural-network-based lossless universal compressors toward multi-source data. *Frontiers of Computer Science*, 19(7):1–16, 2025.
- [28] Xinyu Chen, Jiannan Tian, Ian Beaver, Cynthia Freeman, Yan Yan, Jianguo Wang, and Dingwen Tao. Fcbench: Cross-domain benchmarking of lossless compression for floating-point data. *Proceedings of the VLDB Endowment*, 17(6):1418–1431, 2024.
- [29] Seung Woo Son, Zhengzhang Chen, William Hendrix, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary. Data compression for the exascale computing era-survey. *Supercomputing frontiers and innovations*, 1(2):76–88, 2014.
- [30] Jianguo Wang, Chunbin Lin, Yannis Papakonstantinou, and Steven Swanson. An experimental study of bitmap compression vs. inverted list compression. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 993–1008, 2017.
- [31] Franck Cappello, Sheng Di, Sihuan Li, Xin Liang, Ali Murat Gok, Dingwen Tao, Chun Hong Yoon, Xin-Chuan Wu, Yuri Alexeev, and Frederic T Chong. Use cases of lossy compression for floating-point data in scientific data sets. *The International Journal of High Performance Computing Applications*, 33(6):1201–1220, 2019.
- [32] Ravneet Kaur, Inderveer Chana, and Jhilik Bhattacharya. Data deduplication techniques for efficient cloud storage management: a systematic review. *The Journal of Supercomputing*, 74:2035–2085, 2018.
- [33] Uthayakumar Jayasankar, Vengattaraman Thirumal, and Dhavachelvan Ponnuram. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *Journal of King Saud University-Computer and Information Sciences*, 33(2):119–140, 2021.
- [34] Giacomo Chiarot and Claudio Silvestri. Time series compression survey. *ACM Computing Surveys*, 55(10):1–32, 2023.

- [35] Hui Sun, Huidong Ma, Yingfeng Zheng, Haonan Xie, Meng Yan, Cheng Zhong, Xiaoguang Liu, and Gang Wang. Lrbc: A comprehensive benchmark evaluation of reference-free lossless compression tools for genomics sequencing long reads data. In *2024 Data Compression Conference (DCC)*, pages 584–584. IEEE, 2024.
- [36] Yuansheng Liu, Tao Tang, Zexuan Zhu, Xiangxiang Zeng, Quan Zou, and Keqin Li. Quality scores compression of genomic sequencing data: a comprehensive review and performance evaluation. *IEEE Transactions on Computational Biology and Bioinformatics*, 2025.
- [37] Kelvin V Kredens, Juliano V Martins, Osmar B Dordal, Mauri Ferrandin, Roberto H Herai, Edson E Scalabrin, and Braulio C Avila. Vertical lossless genomic data compression tools for assembled genomes: A systematic literature review. *Plos one*, 15(5):e0232942, 2020.
- [38] M Hernaez, D Pavlichin, T Weissman, et al. Genomic data compression. *Annual Review of Biomedical Data Science*, 2:19–37, 2019.
- [39] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE transactions on visualization and computer graphics*, 12(5):1245–1250, 2006.
- [40] Annie Yang, Hari Mukka, Farbod Hesaaraki, and Martin Burtscher. Mpc: a massively parallel compression algorithm for scientific data. In *2015 IEEE International Conference on Cluster Computing*, pages 381–389. IEEE, 2015.
- [41] Steven Claggett, Sahar Azimi, and Martin Burtscher. Spdp: An automatically synthesized lossless compression algorithm for floating-point data. In *2018 data compression conference*, pages 335–344. IEEE, 2018.
- [42] Chunwei Liu, Hao Jiang, John Paparrizos, and Aaron J Elmore. Decomposed bounded floats for fast compression and queries. *Proceedings of the VLDB Endowment*, 14(11):2586–2598, 2021.
- [43] Fabian Knorr, Peter Thoman, and Thomas Fahringer. Ndzip-gpu: Efficient lossless compression of scientific floating-point data on gpus. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2021.
- [44] Martin Burtscher and Paruj Ratanaworabhan. pfpc: A parallel compressor for floating-point data. In *2009 Data Compression Conference*, pages 43–52. IEEE, 2009.
- [45] Tianyi Zhang, Yang Sui, Shaochen Zhong, Vipin Chaudhary, Xia Hu, and Anshumali Shrivastava. 70% size, 100% accuracy: Lossless llm compression for efficient gpu inference via dynamic-length float. *arXiv preprint arXiv:2504.11651*, 2025.
- [46] Huidong Ma, Hui Sun, Liping Yi, Xiaoguang Liu, and Gang Wang. Multi-source data lossless compression via parallel expansion mapping and xlstm. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2025.
- [47] Yu Mao, Yufei Cui, Tei-Wei Kuo, and Chun Jason Xue. Trace: A fast transformer-based general-purpose lossless compressor. In *Proceedings of the ACM Web Conference 2022*, pages 1829–1838, 2022.
- [48] Chandra Shekhara Kaushik Valmeekam, Krishna Narayanan, Dileep Kalathil, Jean-Francois Chamberland, and Srinivas Shakkottai. Llmzip: Lossless text compression using large language models. *arXiv preprint arXiv:2306.04050*, 2023.
- [49] Byron Knoll. Cmix, 2016. <https://github.com/byronknoll/cmixon>.
- [50] Matt Mahoney. Incremental journaling backup utility and archiver, 2024.
- [51] Igor Pavlov. Lzma official website. <https://tukaani.org/lzma/>, 2013.
- [52] Zstd. Zstd official website, 2024. [Online]. Available: <https://facebook.github.io/zstd/>.
- [53] Ilya Grebnov. Bsc official website. <https://github.com/IlyaGrebnov/libbsc>.
- [54] John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Trans. Commun.*, 32(4):396–402, 1984.

- [55] Alistair Moffat. Implementing the PPM data compression scheme. *IEEE Trans. Commun.*, 38(11):1917–1921, 1990.
- [56] John G. Cleary and W. J. Teahan. Unbounded length contexts for PPM. *Comput. J.*, 40(2/3):67–75, 1997.
- [57] Mark Adler. A parallel implementation of gzip for modern multi-processor, multi-core machines, 2024.
- [58] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [59] Gun Il Kim, Sunga Hwang, and Beakcheol Jang. Efficient compressing and tuning methods for large language models: A systematic literature review. *ACM Computing Surveys*, 2024.
- [60] Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities. *arXiv preprint arXiv:2408.13296*, 2024.
- [61] Seungcheol Park, Jaehyeon Choi, Sojin Lee, and U Kang. A comprehensive survey of compression algorithms for language models. *arXiv preprint arXiv:2401.15347*, 2024.
- [62] Luyang Fang, Xiaowei Yu, Jiazhang Cai, Yongkai Chen, Shushan Wu, Zhengliang Liu, Zhenyuan Yang, Haoran Lu, Xilin Gong, Yufang Liu, et al. Knowledge distillation and dataset distillation of large language models: Emerging trends, challenges, and future directions. *arXiv preprint arXiv:2504.14772*, 2025.
- [63] Amir M Mansourian, Rozhan Ahmadi, Masoud Ghafouri, Amir Mohammad Babaei, Elaheh Badali Golezani, Zeynab Yasamani Ghamchi, Vida Ramezani, Alireza Taherian, Kimia Dinashi, Amirali Miri, et al. A comprehensive survey on knowledge distillation. *arXiv preprint arXiv:2503.12067*, 2025.
- [64] Cunxiang Wang, Xiaoze Liu, Yuanhao Yue, Xiangru Tang, Tianhang Zhang, Cheng Jiayang, Yunzhi Yao, Wenyang Gao, Xuming Hu, Zehan Qi, et al. Survey on factuality in large language models: Knowledge, retrieval and domain-specificity. *arXiv preprint arXiv:2310.07521*, 2023.
- [65] Yanshu Wang, Tong Yang, Xiyang Liang, Guoan Wang, Hanning Lu, Xu Zhe, Yaoming Li, and Li Weitao. Art and science of quantizing large-scale models: A comprehensive overview. *arXiv preprint arXiv:2409.11650*, 2024.
- [66] Ao Shen, Zhiquan Lai, and Dongsheng Li. Exploring quantization techniques for large-scale language models: Methods, challenges and future directions. In *Proceedings of the 2024 9th International Conference on Cyber Security and Information Engineering*, pages 783–790, 2024.
- [67] Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. Keep the cost down: A review on methods to optimize llm’s kv-cache consumption. *arXiv preprint arXiv:2407.18003*, 2024.
- [68] Jiajun Xu, Zhiyuan Li, Wei Chen, Qun Wang, Xin Gao, Qi Cai, and Ziyuan Ling. On-device language models: A comprehensive review. *arXiv preprint arXiv:2409.00088*, 2024.
- [69] Dayou Du, Gu Gong, and Xiaowen Chu. Model quantization and hardware acceleration for vision transformers: A comprehensive survey. *arXiv preprint arXiv:2405.00314*, 2024.
- [70] Zongxia Li, Xiyang Wu, Hongyang Du, Huy Nghiem, and Guangyao Shi. Benchmark evaluations, applications, and challenges of large vision language models: A survey. *arXiv preprint arXiv:2501.02189*, 2025.
- [71] Mikel Hernaez, Dmitri Pavlichin, Tsachy Weissman, and Idoia Ochoa. Genomic data compression. *Annual Review of Biomedical Data Science*, 2:19–37, 2019.
- [72] Mikel Hernaez, Dmitri Pavlichin, Tsachy Weissman, and Idoia Ochoa. Genomic data compression. *Annual Review of Biomedical Data Science*, 2:19–37, 2019.
- [73] R Gilmary, A Venkatesan, and G Vaiyapuri. Compression techniques for dna sequences: A thematic review. *J. Comput. Sci. Eng.*, 15(2):59–71, 2021.

- [74] David A Clunie. Lossless compression of grayscale medical images: effectiveness of traditional and state-of-the-art approaches. *Medical Imaging 2000: PACS Design and Evaluation: Engineering and Clinical Issues*, 3980:74–84, 2000.
- [75] Wilfried Philips, Steven Van Assche, Dirk De Rycke, and Koen Denecker. State-of-the-art techniques for lossless compression of 3d medical image sets. *Computerized Medical Imaging and Graphics*, 25(2):173–185, 2001.
- [76] Neelesh Kumar Sahu and Chandrashekhar Kamargaonkar. A survey on various medical image compression techniques. *International journal of science, Engineering and technology research*, 2(2):501–506, 2012.
- [77] Ashwin Belle, Raghuram Thiagarajan, SM Reza Soroushmehr, Fatemeh Navidi, Daniel A Beard, and Kayvan Najarian. Big data analytics in healthcare. *BioMed research international*, 2015(1):370194, 2015.
- [78] Richard Jumar, Heiko Maaß, and Veit Hagenmeyer. Comparison of lossless compression schemes for high rate electrical grid time series for smart grid monitoring and analysis. *Computers & Electrical Engineering*, 71:465–476, 2018.
- [79] Lulu Wen, Kaile Zhou, Shanlin Yang, and Lanlan Li. Compression of smart meter big data: A survey. *Renewable and Sustainable Energy Reviews*, 91:59–69, 2018.
- [80] Kamil Prokop, Andrzej Bień, and Szymon Barczentewicz. Compression techniques for real-time control and non-time-critical big data in smart grids: A review. *Energies*, 16(24):8077, 2023.
- [81] C Grünler, H Weghorn, and CC Chibelushi. Survey on the state of the art in lossless compression of astronomical images. In *Astronomical Data Analysis Software and Systems XV*, volume 351, page 177, 2006.
- [82] Snigdha Sen, Sonali Agarwal, Pavan Chakraborty, and Krishna Pratap Singh. Astronomical big data processing using machine learning: A comprehensive review. *Experimental Astronomy*, 53(1):1–43, 2022.
- [83] Ce Yu, Bingyao Li, Jian Xiao, Chao Sun, Shanjiang Tang, Chongke Bi, Chenzhou Cui, and Dongwei Fan. Astronomical data fusion: Recent progress and future prospects—a survey. *Experimental Astronomy*, 47:359–380, 2019.
- [84] Evan Nemerson. Scb. <https://quixdb.github.io/squash-benchmark/>.
- [85] Mahoney M. Ltcb. <http://mattmahoney.net/dc/text.html>.
- [86] Hamid Buzidi. Turbobench. <https://github.com/powturbo/TurboBench>.
- [87] Glen G. Langdon Jr. An introduction to arithmetic coding. *IBM J. Res. Dev.*, 28(2):135–149, 1984.
- [88] Jorma Rissanen and Glen G. Langdon Jr. Universal modeling and coding. *IEEE Trans. Inf. Theory*, 27(1):12–22, 1981.
- [89] Lawrence Ibarria, Peter Lindstrom, Jarek Rossignac, and Andrzej Szymczak. Out-of-core compression and decompression of large n-dimensional scalar fields. In *Computer Graphics Forum*, volume 22, pages 343–348. Wiley Online Library, 2003. Issue 3.
- [90] Fabian Knorr, Peter Thoman, and Thomas Fahringer. ndzip: A high-throughput parallel lossless compressor for scientific data. In *2021 Data Compression Conference (DCC)*, pages 103–112. IEEE, 2021.
- [91] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977.
- [92] Solomon Golomb. Run-length encodings (corresp.). *IEEE transactions on information theory*, 12(3):399–401, 1966.
- [93] Dong-Hui Xu, Arati S Kurani, Jacob D Furst, Daniela S Raicu, et al. Run-length encoding for volumetric texture. *Heart*, 27(25):452–458, 2004.

- [94] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
- [95] The HDF Group. HDF5 Filters, 2023. Accessed Feb 13, 2024.
- [96] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraghavan. Gorilla: A fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment*, 8(12):1816–1827, 2015.
- [97] Marek Kokot, Adam Gudyś, Heng Li, and Sebastian Deorowicz. Colord: compressing long reads. *Nature methods*, 19(4):441–444, 2022.
- [98] Hui Sun, Yingfeng Zheng, Haonan Xie, Huidong Ma, Xiaoguang Liu, and Gang Wang. Pmffrc: a large-scale genomic short reads compression optimizer via memory modeling and redundant clustering. *BMC bioinformatics*, 24(1):454, 2023.
- [99] Hui Sun, Huidong Ma, Yingfeng Zheng, Haonan Xie, Xiaofei Wang, Xiaoguang Liu, and Gang Wang. Sr2c: A structurally redundant short reads collapse for optimizing dna data compression. In *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 60–67, 2023.
- [100] Ömer Aydın and Enis Karaarslan. Openai chatgpt generated literature review: Digital twin in healthcare. Aydın, Ö., Karaarslan, E.(2022). *OpenAI ChatGPT Generated Literature Review: Digital Twin in Healthcare*. In Ö. Aydın (Ed.), *Emerging Computer Technologies*, 2:22–31, 2022.
- [101] Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han, and Yang Tang. A brief overview of chatgpt: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica*, 10(5):1122–1136, 2023.
- [102] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [103] Patrick Yubeaton, Tareq Mahmoud, Shehab Naga, Pooria Taheri, Tianhua Xia, Arun George, Yasmeim Khalil, Sai Qian Zhang, Siddharth Joshi, Chinmay Hegde, et al. Huff-llm: End-to-end lossless compression for efficient llm inference. *arXiv preprint arXiv:2502.00922*, 2025.
- [104] Yongchang Hao, Yanshuai Cao, and Lili Mou. Neuzip: Memory-efficient training and inference with dynamic compression of neural networks. *arXiv preprint arXiv:2410.20650*, 2024.
- [105] Khalid Sayood. *Introduction to data compression*. Morgan Kaufmann, 2017.
- [106] Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.
- [107] Hui Sun, Liping Yi, Huidong Ma, Yongxia Sun, Yingfeng Zheng, Wenwen Cui, Meng Yan, Gang Wang, and Xiaoguang Liu. Genomics data lossless compression with (s, k)-mer encoding and deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 12577–12585, 2025. Issue 12.
- [108] Hui Sun, Yanfeng Ding, Liping Yi, Huidong Ma, Haonan Xie, Gang Wang, and Xiaoguang Liu. Adaptive lossless compression for genomics data by multiple (s, k)-mer encoding and xlstm. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2025.
- [109] Hui Sun, Yanfeng Ding, Liping Yi, Huidong Ma, Gang Wang, Xiaoguang Liu, Cheng Zhong, and Wentong Cai. Pmklc: Parallel multi-knowledge learning-based lossless compression for large-scale genomics database. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 2725–2734, 2025.
- [110] Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. Deepzip: Lossless data compression using recurrent neural networks. *arXiv preprint arXiv:1811.08162*, 2018.

- [111] Benyamin Ghogh and Ali Ghodsi. Recurrent neural networks and long short-term memory networks: Tutorial and survey. *arXiv preprint arXiv:2304.11461*, 2023.
- [112] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.
- [113] Qian Liu, Yiling Xu, and Zhu Li. Decmac: A deep context model for high efficiency arithmetic coding. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIIC)*, pages 438–443. IEEE, 2019.
- [114] Hochreiter Sepp and Schmidhuber Jürgen. Long short-term memory. *Neural Computation MIT-Press*, 1997.
- [115] Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, et al. Language modeling is compression. *arXiv preprint arXiv:2309.10668*, 2023.
- [116] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
- [117] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, and Łukasz Kaiser. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.
- [118] Byron Knoll. lstm-compress, 2017. <https://github.com/byronknoll/lstm-compress>.
- [119] F Bellard. Nncp: Lossless data compression with neural networks, 2019. <https://bellard.org/nncp/>.
- [120] Yu Mao, Yufei Cui, Tei-Wei Kuo, and Chun Jason Xue. Accelerating general-purpose lossless compression via simple and scalable parameterization. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 3205–3213, 2022.
- [121] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.
- [122] Yu Mao, Jingzong Li, Yufei Cui, and Chun Jason Xue. Faster and stronger lossless compression with optimized autoregressive framework. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.
- [123] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.
- [124] Huidong Ma, Xinyan Shi, Hui Sun, Xiaofei Yue, Xiaoguang Liu, Gang Wang, and Wentong Cai. Efficient learned data compression via dual-stream feature decoupling, 2026.
- [125] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536, 1978.
- [126] Michael Burrows. A block-sorting lossless data compression algorithm. *SRS Research Report*, 124, 1994.
- [127] Ziya Arnavut, David Leavitt, and Meral Abdulazizoglu. Block sorting transformations. In *Data Compression Conference, DCC 1998, Snowbird, Utah, USA, March 30 - April 1, 1998*, page 524. IEEE Computer Society, 1998.
- [128] Jarek Duda. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*, 2013.
- [129] Huidong Ma. A holistic survey and benchmarking of lossless reference-free compressors for long-read genomics sequencing data. <https://github.com/mhuidong/LRCBench/>.
- [130] Brotli. Brotli official website. <https://github.com/google/brotli>, 2024.
- [131] Yann Collet. Lz4 official website. <https://github.com/lz4/lz4>, 2011.

- [132] Gzip. Gzip official website. <http://www.gzip.org/>, 1996.
- [133] Brieflz. Brieflz official website. <https://github.com/jibsen/brieflz>.
- [134] Lizard. Lizard official website. <https://github.com/inikep/lizard>.
- [135] LZO. Lzo official website. <https://www.oberhumer.com/opensource/lzo/>, 1996.
- [136] Peter Deutsch. Rfc1951: Deflate compressed data format specification version 1.3, 1996.
- [137] Lzop. Lzop official website. <https://www.lzop.org/>.
- [138] Debra A Lelewer and Daniel S Hirschberg. Data compression. *ACM Computing Surveys (CSUR)*, 19(3):261–296, 1987.
- [139] Fast LZMA2. Fast lzma2 official website. <https://github.com/conor42/fast-lzma2>, 2018.
- [140] Kubo Takehiro. Snzip, a compression/decompression tool based on snappy. <https://github.com/kubo/snzip>, 2016.
- [141] Snappy Official Website. Snappy, 2015. <https://github.com/google/snappy>.
- [142] Bzip2. Pbzip2 official website. <https://linux.die.net/man/1/pbzip2>.
- [143] Boris Yakovlevich Ryabko. Data compression by means of a “book stack”. *Problemy Peredachi Informatsii*, 16(4):16–21, 1980.
- [144] PBzip2. pbzip2 - parallel bzip2 file compressor. <https://linux.die.net/man/1/pbzip2>, 2015.
- [145] Bzip2. Bzip2 official website. <https://sourceware.org/bzip2/>, 2024.
- [146] Hui Sun, Yingfeng Zheng, Haonan Xie, Huidong Ma, Cheng Zhong, Meng Yan, Xiaoguang Liu, and Gang Wang. PQSDC: a parallel lossless compressor for quality scores data via sequences partition and Run-Length prediction mapping. *Bioinformatics*, page btae323, 05 2024.
- [147] Yuting Xing, Gen Li, Zhenguo Wang, Bolun Feng, Zhuo Song, and Chengkun Wu. Gtz: a fast compression and cloud transmission tool optimized for fastq files. *BMC bioinformatics*, 18(16):233–242, 2017.
- [148] Eleanor Rieffel and Wolfgang Polak. An introduction to quantum computing for non-physicists. *ACM Computing Surveys (CSUR)*, 32(3):300–335, 2000.
- [149] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.
- [150] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11):4948, 2021.
- [151] ZHONG Cheng and SUN Hui. Parallel algorithm for sensitive sequence recognition from long-read genome data with high error rate. *Journal on Communication/Tongxin Xuebao*, 44(2), 2023.
- [152] Nicholas Wilt. *The cuda handbook: A comprehensive guide to gpu programming*. Pearson Education, 2013.
- [153] 7-Zip. 7-zip official website. <https://www.7-zip.org/>.
- [154] Ipavlov. A free file archiver for extremely high compression, 2024. [Online]. Available: <https://www.7-zip.org>.

A Details of Tested Algorithms

A.1 SPDP

SPDP (Single Precision Double Precision) is a dictionary-based lossless compression algorithm designed for both precision formats. It functions as either an HDF5 filter or an independent compression tool [41]. The execution commands of SPSP are as follows:

```
# compression
./spdp 10 file file.spdp 2
# decompression
./spdp 10 file.spdp file.out 2
```

A.2 FPZip

FPZip is a versatile library and command-line tool designed for lossless and optionally lossy compression of 2D and 3D floating-point arrays. It is optimized for spatially correlated scalar data, such as regularly sampled continuous functions, but is not ideal for compressing unstructured floating-point data streams [39]. The FPZip compression and decompression commands in this manuscript are as follows:

```
# compression and decompression
python fpzip.py
```

A.3 MPC

MPC algorithm [40] is a high-speed, lossless compression method optimized for GPUs and other parallel computing systems. Designed with minimal internal state, MPC achieves exceptional compression ratios across various datasets while significantly outperforming traditional CPU-based algorithms in throughput. Its innovative approach enables efficient real-time data compression, making it ideal for large-scale computing environments. The items reproduce MPC using algorithm scripts from FCBench. The execution commands are as follows:

```
# compression
./MPC_double file 1
# decompression
./MPC_double file.mpc
```

A.4 ZipNN

ZipNN was proposed by Hershcovitch and Wood et al. [22] in 2024. It is an open-source, CPU-based, lossless compressor for AI model tensors that applies type-aware optimizations. The working script and hyperparameters of ZipNN are as follows:

```
# compression
python zipnn_compress_file.py file
# decompression
python zipnn_decompress_file.py file.znn
```

A.5 FM-Delta

FM-Delta is a reference-based lossless compressor specifically designed for large model parameters, published at the NIPS 2024 conference [23]. The working script and hyperparameters are as follows:

```
# compression and decompression
python fmdela.py
```

A.6 MSDZip

MSDZip is the latest neural network-based multimodal data compressor, published at the WWW 2025 conference [26]. Its execution script is as follows:

```
# compression
python compress.py file file.mz --prefix file --gpu
# decompression
python decompress.py file.mz file.mz.out --prefix file --gpu
```

A.7 TRACE

TRACE is a lossless multimodal data compression algorithm built on the Transformer architecture [117], demonstrating strong performance in recompressing Large Language Models [47]. The execution script of TRACE is as follows:

```
# compression
python compressor.py --source file --comp file.trace
# decompression
python compressor.py --comp file.trace --decomp file.trace.out
```

A.8 PAC

PAC is a lossless multimodal data compression algorithm proposed by Mao et al. [122], developed by the same team behind TRACE. It utilizes a context-based modeling approach, with its prediction component built upon a multilayer perceptron (MLP) [121]. The execution commands for PAC are as follows:

```
# compression
python compressor.py --source file --comp file.pac
# decompression
python compressor.py --comp file.pac --decomp file.pac.out
```

A.9 DeepZip

DeepZip is a static compression algorithm introduced by Goyal et al. [110] in 2019, built upon an RNN-based architecture. Its compression process necessitates storing the trained model within the final output file. The execution script for DeepZip is as follows:

```
#compression
sh./compress.sh file file.deepzip bs model
#decompression
sh./decompress.sh file.deepzip file.deepzip.out bs model
```

A.10 DZip

DZip, developed by the same team behind DeepZip, is an integrated compression solution based on BiGRU and MLP. In this work, DZip introduces a pioneering semi-adaptive lossless compression framework. The execution commands for DZip are as follows:

```
# compression
sh ./compress.sh file file.dzip com model
# decompression
sh ./decompress.sh file.dzip file.dzip.out com model
```

A.11 LZMA

LZMA (Lempel-Ziv-Markov chain algorithm) [51] is an efficient data compression algorithm, originally developed by Igor Pavlov and implemented in the 7-Zip compression tool. It uses a dictionary coding mechanism, similar to the LZ77 algorithm [91], but generally achieves a higher compression ratio than Bzip2. The working script and hyperparameters of LZMA are as follows:

```
# compression
lzma a -m0=lzma -mx9 -mmt16 file.7z file
# decompression
lzma x -y -mx9 -mmt file.7z
```

A.12 LZMA2

LZMA2 Algorithm [139] improves the multi-threading capability and performance of the LZMA algorithm and better handles incompressible data, so the compression performance is slightly improved. We also used the built-in Lzma2 algorithm in the 7-Zip application [153].

```
# compression
7zz a -m0=lzma2 -mx9 -mmt16 file.7z file
# decompression
7zz x -y -mx9 -mmt16 file.7z
```

A.13 PPMD

PPMD [54; 55; 56] is a context-based compression method, built upon the Partial Matching Prediction (PPM) algorithm introduced by Cleary and Witten. PPM employs statistical modeling by analyzing prior symbols in the input sequence to predict the next symbol, thereby reducing the entropy of the output data. Unlike dictionary-based compression, which encodes symbols by locating them within a predefined dictionary, PPM focuses on probabilistically predicting upcoming symbols. In our implementation, we utilize PPMD within 7-Zip [154] to perform data compression.

```
# Compression
7zz a -m0=ppmd -mx9 -mmt16 file.7z file
# Decompression
7zz x -y -mx9 -mmt16 file.7z
```

A.14 LZOP

LZOP is a file compression tool that closely resembles GZip [132]. It relies on the LZO data compression library to perform compression, offering significantly faster compression and decompression speeds compared to gzip, though with a slight trade-off in compression ratio [137].

```
# Compression
lzop -9 -f file -o file.lzop
# Decompression
lzop -df file.lzop -o file.lzop.out
```

A.15 BZip2

Bzip2 is an open-source, patent-free data compression tool known for its high efficiency. It generally achieves file compression rates within 10% to 15% of the most advanced statistical compressors from the PPM family while offering approximately twice the compression speed and six times the decompression speed [145].

```
# Compression
bzip2 -kzfc9 file > file.bz2
# Decompression
bzip2 -cdf file.bz2 > file.bz2.out
```

A.16 PBzip2

PBzip2 is a parallelized variant of the BZip2 [145] block-sorting compression algorithm, designed to leverage pthreads for near-linear speedup on SMP systems [144].

```
# Compression
pbzip2 -9 -m2000 -p16 -c file > file.bz2
# Decompression
pbzip2 -dc -9 -p16 -m2000 file.bz2
```

A.17 Lizard

Lizard is a high-performance compression tool designed for rapid decompression. At low to medium compression levels, it delivers a compression ratio comparable to Zip/Zlib and Zstd/Brotli while maintaining exceptional

decompression speed [134].

```
# Compression
lizard -49 -f file file.lizard -BD
# Decompression
lizard -df file.lizard file.lizard.out
```

A.18 BSC

BSC is a high-performance file compressor utilizing a lossless block-ordered data compression algorithm. This manuscript employs BSC V3.3.2 for both compression and decompression tasks [53].

```
# Compression
bsc e file file.bsc -e2
# Decompression
bsc d file.bsc file.bsc.out
```

A.19 Brotli

Brotli is a versatile, lossless compression algorithm that integrates an advanced variant of LZ77, Huffman coding, and second-order context modeling. It delivers a compression ratio comparable to top-tier general-purpose methods while maintaining a speed similar to deflate but achieving more compact compression [130].

```
# Compression
brotli file -o file.brotli
# Decompression
brotli -d file.brotli -o file.brotli.out
```

A.20 LZO

LZO is a portable lossless data compression library written in ANSI C. LZO Offers pretty fast compression and extremely fast decompression [135].

```
# Compression
lzo -9 file file.lzo
# Decompression
lzo -d file.lzo file.lzo.out
```

A.21 PIGZ

PIGZ leverages multithreading to efficiently utilize multiple processors and cores for compression. It divides the input into 128 KB chunks, compressing each in parallel while simultaneously calculating individual check values. The compressed data is then sequentially written to the output, with a final combined check value derived from the individual calculations [57].

```
# Compression
pigz -cf -9 -p16 file > file.pigz
# Decompression
pigz -dfc -p16 file.pigz > file.pigz.out
```

A.22 SnZip

SnZip is a general-purpose, lossless compression algorithm built on snappy. It supports multiple file formats, including framing-format and old framing-format, with framing-format set as the default. The following command lines demonstrate how to use SnZip for compression and decompression [140].

```
# compression
snzip -k -t snzip file
# decompression
snzip -kd -t snzip file.snz
```

A.23 ZSTD

ZSTD is a high-speed, lossless compression algorithm designed for real-time applications, offering compression ratios that surpass zlib-level efficiency. It leverages an advanced entropy stage powered by Huff0 and the FSE library to enhance performance [52].

```
# Compression
zstd -19 -f file -o file.zstd
# Decompression
zstd -df file.zstd -o file.zstd.out
```

A.24 ZPAQ

ZPAQ is a free, open-source command-line archiver designed for Windows, Linux, and macOS. It supports incremental backups, meaning only files that have changed since the last backup are added, optimizing storage and backup speed. Additionally, its journaling feature ensures that previous versions remain accessible, allowing rollback to earlier states when needed [50].

```
# Compression
zpaq a file.zpaq file -t16 -method 5
# Decompression
zpaq x file.zpaq -method 5 -t 16
```

B Additional Results

Table 6: Experimental Results Obtained by Running DeepZip on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Hours)	CPM (MegaBytes)	DT (Hours)	DPM (MegaBytes)
D_0	10304.880	1.515	34.039	424.681	254205.649	45.661	157622.792
D_1	9575.263	1.516	34.080	394.873	236373.480	42.001	146616.540
D_2	1128.736	1.199	16.611	36.781	89363.719	5.213	14371.363
D_3	1062.526	1.200	16.635	34.591	84228.199	4.902	13582.602
D_4	980.554	1.205	16.980	31.971	77929.996	4.395	12648.289
D_5	623.685	1.511	33.814	25.485	62249.426	3.373	10262.445
D_6	526.029	1.281	21.929	18.312	45088.262	2.571	7577.199
D_7	174.679	2.938	65.960	13.549	34626.488	1.597	5975.504
D_8	217.446	1.175	14.876	6.918	17843.809	0.983	3400.824
D_9	77.513	1.086	7.934	2.293	6689.785	0.335	1686.348

Table 7: Experimental Results Obtained by Running DZip on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Hours)	CPM (MegaBytes)	DT (Hours)	DPM (MegaBytes)
D_0	10305.166	1.516	34.037	584.531	293406.300	181.924	157749.871
D_1	9575.262	1.517	34.080	542.964	272850.785	168.947	146764.127
D_2	1128.731	1.199	16.611	49.982	26103.855	17.821	14757.313
D_3	1062.457	1.200	16.640	48.747	24603.230	16.655	13999.555
D_4	980.547	1.205	16.981	43.274	22826.652	15.401	13003.625
D_5	623.671	1.511	33.816	34.588	18289.094	11.959	10644.590
D_6	525.932	1.281	21.944	24.891	13187.078	8.716	7927.164
D_7	174.667	2.938	65.963	18.669	10138.887	6.489	6339.023
D_8	217.444	1.175	14.876	9.372	5265.609	3.364	3755.379
D_9	77.512	1.086	7.935	3.121	2530.395	1.135	2052.535

Table 8: Experimental Results Obtained by Running PAC on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Hours)	CPM (MegaBytes)	DT (Hours)	DPM (MegaBytes)
D_0	10234.953	1.526	34.486	142.111	265944.756	181.726	266626.138
D_1	9500.113	1.529	34.598	134.014	247298.700	168.917	245649.535
D_2	1120.526	1.208	17.217	12.287	23395.535	15.542	24079.910
D_3	1054.166	1.209	17.291	11.526	22052.973	15.051	22721.988
D_4	973.124	1.214	17.609	11.389	20464.078	13.743	21143.766
D_5	616.300	1.529	34.598	8.426	16405.152	10.619	17063.484
D_6	522.253	1.290	22.490	6.067	11840.211	7.553	12516.703
D_7	166.683	3.079	67.519	4.083	9109.898	5.103	9776.340
D_8	209.893	1.217	17.833	2.484	4728.676	3.003	5391.102
D_9	69.858	1.205	17.026	0.841	1815.543	1.004	2485.484

Table 9: Experimental Results Obtained by Running TRACE on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Hours)	CPM (MegaBytes)	DT (Hours)	DPM (MegaBytes)
D_0	10279.142	1.520	34.204	136.155	265942.601	181.613	265206.588
D_1	9530.574	1.524	34.388	126.836	247318.511	168.613	245661.007
D_2	1122.250	1.206	17.090	10.170	23396.035	14.578	24116.273
D_3	1055.829	1.207	17.160	9.505	22053.219	14.141	22776.848
D_4	979.953	1.205	17.031	8.891	20465.324	15.171	21182.957
D_5	618.912	1.523	34.321	7.098	16404.906	10.468	17127.316
D_6	525.129	1.283	22.063	5.019	11840.672	7.436	12563.965
D_7	167.605	3.062	67.339	3.539	9109.961	4.633	9829.938
D_8	210.564	1.213	17.570	1.929	4728.348	2.724	5450.996
D_9	70.037	1.202	16.813	0.634	1816.645	0.919	2537.750

Table 10: Experimental Results Obtained by Running MSDZIP on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Hours)	CPM (MegaBytes)	DT (Hours)	DPM (MegaBytes)
D_0	10235.047	1.526	34.486	276.428	265970.456	272.879	265852.720
D_1	9500.117	1.529	34.598	253.361	247323.905	253.932	247603.148
D_2	1120.435	1.208	17.224	24.476	23395.863	24.359	24237.234
D_3	1054.089	1.209	17.297	22.853	22051.555	23.777	22878.762
D_4	973.150	1.214	17.607	24.203	20462.520	21.802	21292.773
D_5	616.142	1.529	34.615	15.086	16402.641	16.477	17230.957
D_6	521.600	1.292	22.587	14.694	11838.730	13.038	12668.668
D_7	166.663	3.079	67.522	8.089	9108.039	8.698	9945.734
D_8	209.873	1.217	17.841	4.308	4727.117	4.575	5557.809
D_9	69.851	1.205	17.034	1.433	1892.922	1.533	2646.789

Table 11: Experimental Results Obtained by Running LZMA on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	11050.576	1.414	29.266	11451.001	685.668	704.058	72042.000
D_1	10259.512	1.416	29.370	10449.001	685.905	655.06	72312.000
D_2	1223.331	1.106	9.622	390.016	685.004	51.099	71912.000
D_3	1150.286	1.108	9.749	360.091	685.055	49.013	72012.000
D_4	1060.260	1.114	10.232	332.084	685.164	45.031	72032.000
D_5	660.084	1.428	29.951	572.037	685.066	29.022	72064.000
D_6	569.954	1.182	15.410	233.034	685.086	24.052	71956.000
D_7	180.981	2.835	64.732	446.073	685.102	9.063	72148.000
D_8	229.287	1.114	10.240	67.014	685.063	10.051	72168.000
D_9	76.885	1.095	8.680	16.087	671.102	3.098	72044.000

Table 12: Experimental Results Obtained by Running LZMA2 on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	10979.638	1.423	29.720	1435.053	263.105	1457.037	262.220
D_1	10196.666	1.425	29.802	1187.048	262.908	1352.245	262.091
D_2	1239.498	1.092	8.428	883.004	262.988	952.037	262.320
D_3	1165.229	1.094	8.577	792.099	262.984	843.099	262.344
D_4	1074.658	1.099	9.013	718.009	263.109	763.029	262.207
D_5	648.420	1.453	31.189	570.064	263.063	603.081	262.289
D_6	577.833	1.166	14.241	428.067	263.152	452.063	262.199
D_7	175.008	2.932	65.896	163.078	263.164	171.004	262.285
D_8	232.003	1.101	9.177	172.091	263.164	182.037	262.297
D_9	77.524	1.086	7.921	58.008	263.387	60.097	262.273

Table 13: Experimental Results Obtained by Running PPMD on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	10773.873	1.450	31.037	11468.001	263.363	13926.001	262.277
D_1	9999.938	1.453	31.157	8667.001	263.223	8616.001	262.273
D_2	10979.638	1.423	29.720	1435.053	263.105	1457.037	262.220
D_3	1274.563	1.000	-0.001	85.092	2.391	2.029	1.641
D_4	1177.976	1.003	0.265	79.071	2.391	2.020	1.645
D_5	880.838	1.070	6.525	131.047	2.332	4.065	1.625
D_6	646.764	1.042	4.010	60.008	2.371	1.097	1.672
D_7	260.249	1.972	49.285	378.071	2.395	2.043	1.625
D_8	255.448	1.000	-0.001	18.025	2.367	1.026	1.629
D_9	84.176	1.000	0.020	6.067	2.355	1.009	1.680

Table 14: Experimental Results Obtained by Running LZOP on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	14674.325	1.065	6.070	2312.037	2.539	98.051	1.730
D_1	13616.000	1.067	6.263	2110.082	2.691	127.043	1.879
D_2	1353.610	1.000	-0.003	95.002	2.402	2.039	1.629
D_3	1274.563	1.000	-0.001	85.092	2.391	2.029	1.641
D_4	1177.976	1.003	0.265	79.071	2.391	2.020	1.645
D_5	880.838	1.070	6.525	131.047	2.332	4.065	1.625
D_6	646.764	1.042	4.010	60.008	2.371	1.097	1.672
D_7	260.249	1.972	49.285	378.071	2.395	2.043	1.625
D_8	255.448	1.000	-0.001	18.025	2.367	1.026	1.629
D_9	84.176	1.000	0.020	6.067	2.355	1.009	1.680

Table 15: Experimental Results Obtained by Running BZip2 on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	10836.847	1.442	30.634	2016.087	8.641	904.024	4.625
D_1	10067.848	1.443	30.689	1686.004	7.730	786.076	4.746
D_2	1284.675	1.054	5.090	158.057	7.699	82.058	4.539
D_3	1208.456	1.055	5.185	147.067	7.391	82.004	4.805
D_4	1117.469	1.057	5.388	136.095	7.465	72.026	4.566
D_5	654.186	1.440	30.577	95.017	7.465	49.052	4.566
D_6	599.686	1.124	10.998	76.022	7.398	40.030	4.563
D_7	176.419	2.909	65.621	42.005	7.418	19.024	4.629
D_8	241.390	1.058	5.502	30.024	7.391	16.030	4.684
D_9	80.119	1.051	4.839	10.077	7.402	6.016	4.629

Table 16: Experimental Results Obtained by Running PBZip2 on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	10837.760	1.442	30.628	171.041	1546.418	191.025	2027.598
D_1	10068.745	1.443	30.683	97.041	153.668	54.013	109.008
D_2	1284.756	1.054	5.084	11.063	160.094	7.024	115.781
D_3	1208.534	1.055	5.179	11.002	159.930	6.088	115.582
D_4	1117.542	1.057	5.382	10.016	0.010	6.068	115.609
D_5	654.241	1.440	30.571	7.074	152.785	5.003	108.477
D_6	599.725	1.123	10.992	6.026	160.031	4.026	115.508
D_7	176.447	2.908	65.616	4.024	143.004	3.017	97.762
D_8	241.404	1.058	5.497	3.023	159.648	2.045	115.633
D_9	80.123	1.051	4.833	1.089	156.469	1.052	114.633

Table 17: Experimental Results Obtained by Running Lizard on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	15585.733	1.002	0.236	5193.001	174.289	25.049	41.875
D_1	14452.705	1.005	0.502	4075.001	174.191	30.041	41.910
D_2	1353.549	1.000	0.002	210.034	173.832	2.078	41.633
D_3	1274.504	1.000	0.003	197.097	173.957	2.065	41.664
D_4	1173.898	1.006	0.611	185.008	173.867	2.060	41.688
D_5	931.735	1.011	1.124	261.097	173.824	2.036	41.902
D_6	638.306	1.056	5.266	139.081	174.219	2.009	41.621
D_7	219.828	2.334	57.162	281.034	171.512	2.023	39.367
D_8	255.436	1.000	0.004	40.089	173.844	1.038	41.750
D_9	84.171	1.000	0.026	13.081	173.848	1.017	41.801

Table 18: Experimental Results Obtained by Running BSC on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	10402.416	1.502	33.414	121.049	11730.039	75.064	6989.559
D_1	9654.963	1.504	33.532	118.093	11859.797	72.007	6447.070
D_2	1183.735	1.143	12.547	22.002	8007.629	12.043	3657.746
D_3	1112.350	1.146	12.726	20.005	7507.965	12.063	3674.605
D_4	1027.102	1.150	13.039	18.047	6928.461	12.045	3598.594
D_5	626.335	1.505	33.533	11.082	5529.820	8.023	3492.125
D_6	553.065	1.218	17.917	11.039	3901.902	8.089	2803.414
D_7	169.722	3.024	66.926	5.031	2963.590	4.011	2088.203
D_8	221.880	1.151	13.140	9.037	1422.738	7.061	1270.703
D_9	74.207	1.135	11.861	8.047	430.164	7.028	397.281

Table 19: Experimental Results Obtained by Running Brotli on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	11198.356	1.395	28.320	55821.001	211.520	194.040	18.938
D_1	10391.806	1.398	28.459	48296.001	234.121	183.002	19.051
D_2	1241.607	1.090	8.272	3867.001	202.680	16.087	19.000
D_3	1168.729	1.091	8.302	3641.001	202.574	15.068	19.109
D_4	1077.035	1.097	8.812	3318.050	207.289	14.079	19.078
D_5	673.971	1.398	28.478	2872.011	205.063	12.097	19.109
D_6	579.131	1.163	14.048	1949.094	278.359	9.021	19.031
D_7	187.117	2.742	63.536	1204.019	257.719	3.087	19.035
D_8	232.817	1.097	8.858	725.015	202.547	3.098	19.074
D_9	77.343	1.089	8.136	217.038	202.574	2.004	19.051

Table 20: Experimental Results Obtained by Running LZO on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	14674.087	1.065	6.072	2220.037	2.344	72.041	1.633
D_1	13615.771	1.067	6.264	1985.053	2.121	112.033	1.652
D_2	1353.590	1.000	-0.001	105.069	2.164	2.036	1.656
D_3	1274.544	1.000	0.000	96.025	2.355	2.040	1.668
D_4	1177.958	1.003	0.267	88.044	2.336	2.030	1.641
D_5	880.824	1.070	6.526	124.058	2.164	4.078	1.672
D_6	646.754	1.042	4.012	64.046	2.168	1.099	1.766
D_7	261.309	1.964	49.079	284.006	2.156	2.044	1.766
D_8	255.444	1.000	0.000	19.012	2.168	1.028	1.672
D_9	84.174	1.000	0.022	7.009	2.125	1.009	1.609

Table 21: Experimental Results Obtained by Running PIGZ on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	12455.639	1.254	20.272	167.022	14.637	150.071	2.391
D_1	11564.542	1.256	20.385	85.092	14.734	82.096	2.535
D_2	1257.830	1.076	7.073	5.009	15.266	7.003	2.449
D_3	1183.400	1.077	7.151	4.085	14.773	6.070	2.563
D_4	1092.601	1.081	7.494	4.058	15.117	6.032	2.434
D_5	747.982	1.260	20.624	6.058	14.023	6.026	2.340
D_6	598.134	1.126	11.228	4.064	15.180	4.020	2.484
D_7	226.682	2.264	55.826	36.000	12.656	3.003	2.363
D_8	235.577	1.084	7.778	1.083	14.098	2.016	2.309
D_9	78.186	1.077	7.135	1.029	13.688	1.042	2.418

Table 22: Experimental Results Obtained by Running Snzip on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	15622.295	1.000	0.002	256.015	3.242	199.008	3.105
D_1	14518.026	1.001	0.053	12.078	3.387	11.075	3.379
D_2	1353.735	1.000	-0.012	2.007	3.449	1.097	3.258
D_3	1274.682	1.000	-0.011	2.004	3.262	1.091	3.133
D_4	1180.973	1.000	0.012	1.095	3.320	1.084	3.133
D_5	941.873	1.000	0.048	1.077	3.270	1.066	3.133
D_6	672.535	1.002	0.186	1.078	3.266	1.058	3.199
D_7	356.209	1.441	30.585	8.051	3.273	5.055	3.379
D_8	255.471	1.000	-0.010	1.023	3.270	1.021	3.258
D_9	84.184	1.000	0.011	1.009	3.270	1.008	3.133

Table 23: Experimental Results Obtained by Running ZSTD on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	11861.774	1.317	24.073	9360.001	265.449	102.020	13.309
D_1	11001.427	1.320	24.262	7329.001	264.180	67.025	13.414
D_2	1255.875	1.078	7.218	429.074	272.875	2.089	13.016
D_3	1181.509	1.079	7.300	378.036	272.883	2.049	12.488
D_4	1087.272	1.086	7.945	341.040	273.188	2.039	12.617
D_5	713.028	1.322	24.333	444.017	262.879	4.064	12.594
D_6	589.123	1.144	12.565	242.097	273.508	1.098	12.277
D_7	201.799	2.543	60.675	374.067	238.855	2.034	13.098
D_8	235.118	1.086	7.958	75.041	272.875	1.031	13.008
D_9	78.077	1.078	7.264	24.083	228.598	1.014	12.227

Table 24: Experimental Results Obtained by Running ZPAQ on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	10269.523	1.521	34.265	4395.001	14044.047	4529.001	11454.223
D_1	9530.586	1.524	34.388	4881.001	14090.086	4777.001	11469.902
D_2	1124.051	1.204	16.957	568.018	13087.289	588.002	11901.344
D_3	1057.712	1.205	17.013	562.038	13051.684	561.074	11985.613
D_4	975.176	1.211	17.436	541.014	12876.594	551.096	11974.086
D_5	618.214	1.524	34.395	272.038	11011.016	264.059	10417.055
D_6	525.486	1.282	22.010	285.088	8210.523	299.064	7628.082
D_7	167.264	3.068	67.405	178.065	5243.508	178.029	5040.164
D_8	210.661	1.213	17.532	260.041	3212.211	262.057	3003.844
D_9	70.108	1.201	16.729	243.034	1183.359	244.035	1106.754

Table 25: Experimental Results Obtained by Running ZipNN on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	10418.782	1.499	33.310	67.123	52516.965	25.722	26465.961
D_1	9674.274	1.501	33.399	34.620	48834.598	20.711	24625.793
D_2	1201.308	1.127	11.249	4.895	5504.418	3.653	2972.387
D_3	1131.047	1.127	11.259	4.833	5213.492	3.777	2823.527
D_4	1048.611	1.126	11.218	4.563	4861.184	3.356	2646.832
D_5	627.736	1.501	33.384	3.750	3517.348	2.845	1987.340
D_6	571.835	1.178	15.131	3.246	2885.613	2.739	1663.801
D_7	232.992	2.202	54.597	2.813	1894.320	2.299	1165.836
D_8	226.462	1.128	11.346	2.358	1349.789	2.163	899.539
D_9	74.913	1.124	11.021	2.042	695.777	1.889	576.074

Table 26: Experimental Results Obtained by Running FM-Delta on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	9349.271	1.671	40.156	235.215	14162.855	30.412	12250.572
D_1	8674.633	1.674	40.281	218.456	12905.725	28.154	10991.819
D_2	1079.632	1.254	20.238	22.767	1270.387	23.505	1344.387
D_3	869.666	1.466	31.767	20.788	946.629	20.871	1285.051
D_4	1029.047	1.148	12.875	15.497	887.906	15.747	876.273
D_5	575.581	1.637	38.919	32.304	1415.828	31.885	1863.391
D_6	495.445	1.360	26.469	10.119	543.406	10.595	687.840
D_7	421.151	1.218	17.930	8.450	457.852	8.902	517.234
D_8	37.461	6.819	85.335	1.653	61.102	2.027	269.090
D_9	59.962	1.404	28.780	1.302	67.910	1.390	88.195

Table 27: Experimental Results Obtained by Running FPZip on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	13641.837	1.145	12.679	345.124	15526.127	351.125	14228.394
D_1	12532.936	1.159	13.719	278.154	14118.041	254.875	14252.570
D_2	1231.328	1.099	9.031	27.050	1412.184	24.303	1327.570
D_3	1178.678	1.081	7.522	25.005	1282.531	22.696	1283.941
D_4	1061.703	1.112	10.110	24.835	1279.641	21.149	1181.215
D_5	724.986	1.300	23.064	39.174	1821.336	33.987	1860.613
D_6	623.973	1.080	7.393	13.052	674.160	11.901	678.617
D_7	448.396	1.144	12.621	9.505	509.160	8.718	526.156
D_8	232.686	1.098	8.910	5.028	274.191	4.492	250.891
D_9	77.931	1.080	7.438	1.484	93.035	1.426	91.902

Table 28: Experimental Results Obtained by Running MPC on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	1216.218	1.113	10.148	2.853	2669.906	2.923	2670.191
D_1	1145.345	1.113	10.137	2.952	2520.246	2.966	2520.207
D_2	1061.378	1.113	10.137	2.556	2342.730	2.651	2342.934
D_3	737.539	1.278	21.732	2.016	1780.297	2.092	1780.035
D_4	574.777	1.172	14.694	1.484	1348.938	1.525	1348.930
D_5	219.715	2.336	57.184	0.908	833.031	1.077	833.199
D_6	229.379	1.114	10.204	0.855	585.305	0.886	585.141
D_7	76.424	1.102	9.227	0.538	259.746	0.380	260.441

Table 29: Experimental Results Obtained by Running SPDP on Benchmark Datasets

ID	CS (MegaBytes)	CR (Ratio)	SSP (%)	CT (Seconds)	CPM (MegaBytes)	DT (Seconds)	DPM (MegaBytes)
D_0	15752.021	0.992	-0.828	117.938	18.797	105.281	18.680
D_1	14655.872	0.991	-0.896	103.553	18.781	98.819	18.652
D_2	1360.904	0.995	-0.542	7.637	18.637	7.875	18.707
D_3	1281.766	0.994	-0.567	7.326	18.730	7.400	18.703
D_4	1180.478	1.001	0.054	8.981	18.691	10.367	18.754
D_5	948.053	0.994	-0.608	5.792	18.785	5.839	18.754
D_6	652.147	1.033	3.212	3.894	18.707	3.889	18.703
D_7	262.136	1.958	48.917	2.037	18.695	2.184	18.680
D_8	257.742	0.991	-0.899	1.562	18.863	1.604	18.832
D_9	84.573	0.996	-0.451	0.525	18.691	0.530	18.727

Table 30: The Overall Performance of Tested Algorithms on All Benchmark Datasets

Method	AvgCR (Ratio)	WavgCR (Ratio)	AvgSSP (%)	WAvgSSP (%)	CRP (%)	TotalCT (Hours)	TotalDT (Hours)	AvgCPM (GB)	AvgDPM (GB)
MSDZip	1.501	1.509	28.081	32.832	36.235	644.931	641.069	76.071	76.894
PAC	1.501	1.509	28.067	32.829	36.240	333.228	422.260	76.057	76.594
ZPAQ	1.495	1.504	27.813	32.607	36.244	3.385	3.404	11.720	10.496
TRACE	1.495	1.503	27.798	32.576	36.140	309.777	420.296	76.059	76.471
DZip	1.463	1.495	26.283	32.250	35.178	1360.140	432.411	84.131	46.020
DeepZip	1.463	1.495	26.281	32.250	35.176	989.452	111.031	110.913	45.623
BSC	1.448	1.479	24.864	31.298	37.855	0.095	0.060	7.359	4.202
PPMD	1.393	1.426	21.664	28.722	38.516	6.644	7.380	0.321	0.320
LZMA	1.381	1.396	21.725	27.357	36.442	6.755	0.439	0.691	0.071
LZMA2	1.379	1.403	21.671	27.687	36.025	0.961	0.049	5.344	3.132
BZip2	1.363	1.413	19.452	27.890	39.777	1.221	0.571	0.009	0.006
PBZip2	1.363	1.413	19.447	27.884	39.772	0.089	0.078	0.341	0.371
ZipNN	1.351	1.462	22.591	30.798	24.254	0.036	0.019	15.536	8.036
Brotli	1.346	1.281	18.555	16.572	39.901	4.943	0.021	1.718	0.149
ZSTD	1.305	1.306	18.359	22.570	32.583	5.277	0.051	0.320	0.016
MPC	1.292	1.244	17.933	16.205	30.803	0.004	0.004	12.051	12.052
PIGZ	1.256	1.247	16.497	19.178	27.463	0.087	0.074	0.018	0.003
Lizard	1.142	1.023	6.494	1.253	34.861	2.943	0.019	0.212	0.051
LZOP	1.136	1.095	7.513	5.487	27.904	0.237	0.004	0.019	0.013
FPZip	1.130	1.148	11.249	12.826	5.579	0.213	0.204	4.515	4.356
LZO	1.121	1.071	7.224	6.045	25.198	1.387	0.055	0.003	0.002
FM-Delta	1.120	1.006	4.739	-0.025	26.316	0.072	0.068	0.023	0.023
SPDP	1.120	1.006	4.739	-0.025	26.316	0.072	0.068	0.023	0.023
SnZip	1.044	1.006	3.086	0.457	12.649	0.079	0.062	0.004	0.004
Brieflz	1.041	0.977	-2.393	-3.338	33.995	1.184	0.105	0.029	0.004

NOTE: The colors of “Green”, “Pink”, and “Light Blue” respectively represent the Top 1-3 performance. “AvgCR”, “WavgCR”, “AvgSSP”, “WavgSSP” denote the average / weighted average compression ratio, average / weighted average storage saving percentage, and compression robustness performance, respectively. “TotalCT” and “TotalDT” denote the total time for compression and decompression, respectively. “AvgCPM” and “AvgDPM” denote average CPU compression / decompression peak memory usage, respectively.