nature machine intelligence

Supplementary information

https://doi.org/10.1038/s42256-025-01033-7

Lossless data compression by large models

In the format provided by the authors and unedited

S1. Experiment Setup

This section provides a comprehensive description of our experimental setup, encompassing hardware, software, various hyperparameters, and other factors that could potentially influence the outcomes of our experiments.

S1.1 Image

Hardware The image compression experiments are conducted on a computational node equipped with a single NVIDIA A100 GPU (40GB), six Intel(R) Xeon(R) Gold 6248R CPUs running at 3.00GHz, and 45GB of RAM.
Software Image compression, including PNG, JPEG-XL, and other traditional baselines, is facilitated by the Python Imaging Library (PIL), version 10.2.0.
Baselines We compare LMCompress against several lossless image compression methods, including PNG [1], WebP [2], JPEG-XL [3], and JPEG-2000 [4]. PNG is among the most widely supported lossless formats. JPEG-XL is a transform-based approach designed to offer high-quality lossless compression along with advanced features, such as support for high dynamic range (HDR) and wide color gamuts. WebP is a modern image format developed by Google, designed to provide superior compression methods while maintaining high visual quality. JPEG-2000 is an image compression standard that was developed by the Joint Photographic Experts Group (JPEG) as an enhancement over the original JPEG format. It was designed to offer both lossy and lossless compression, with a focus on high-quality image storage and more efficient compression algorithms.

S1.2 Vedio

Hardware The video compression experiments are conducted on a computational node equipped with a single NVIDIA A100 GPU (40GB), six Intel(R) Xeon(R) Gold 6248R CPUs running at 3.00GHz, and 45GB of RAM.
Software The operating system is KylinOS V10. All the video compression methods we used are from FFmpeg n7.1-16-g15035aaec0.
Parameters In the experiment, the parameter configuration in ffmpeg is crf:0 to ensure lossless compression.
Baselines We compare LMCompress against several lossless image compression methods, including H.264 [5], H.264 [6] and FFV1 [7]. H.264 is a widely adopted video compression standard developed by the ITU-T Video Coding Experts Group and ISO/IEC Moving Picture Experts Group. It is the most commonly used video codec for a range of applications, including video streaming, broadcasting, video conferencing, and video storage. H.265 is the successor to H.264 and was designed to offer improved compression efficiency, reducing the bit rate required for delivering high-quality video. FFV1 is a lossless video codec developed as part of the FFmpeg project, designed primarily for archiving and preservation of high-quality video content. It is optimized for retaining the original quality of the video while minimizing storage requirements.

S1.3 Audio

Hardware The audio compression experiments are conducted on a computational node equipped with a single NVIDIA A100 GPU (40GB), six Intel(R) Xeon(R) Gold 6248R CPUs running at 3.00GHz, and 45GB of RAM.

Software The operating system is KylinOS V10. The FLAC implementation is obtained from the FFmpeg 20240609 static build, while the OptimFROG and WavPack implementations are versions 5.10 and 5.70, respectively.

Parameters The parameters are set to provide the best compression rates for each lossless audio codec, including the baselines and our proposed method. Table 1 summarizes the parameter configurations of the baselines.

Baselines The baselines include several traditional lossless audio compression methods and a recent large-model-based approach. The traditional baselines include FLAC [8], WavPack [9], and OptimFROG [10]. FLAC is one of the most widely used open-source lossless audio codecs. WavPack is another open-source codec supporting both lossless and hybrid compression, while we only study its lossless mode. OptimFROG is a high-compression lossless codec designed to achieve superior compression ratios, often at the cost of higher computational requirements. The large-model-based baseline is adapted from the work of a DeepMind team [11].

Program	Parameters
FLAC	`--compression_level 12`
OptimFROG	`--preset max`
WavPack	`-hh -x3`

Table 1 Configurations for the audio compression baselines.

Potential Speedup The WavPack implementation used in our experiments supports an explicit multi-threading option, however, the number of threads is fixed to 1 for a fair comparison. Similarly, the large model compressors, including the method proposed in [11] and our approach, are theoretically capable of achieving faster compression when using a batch size greater than 1. Nevertheless, for consistency and fair comparison, the batch size is also set to 1.

S1.4 Text

Hardware The text compression experiments are conducted on a computational node equipped with a single NVIDIA A100 GPU (40GB), six Intel(R) Xeon(R) Gold 6248R CPUs running at 3.00GHz, and 45GB of RAM.

Software The operating system used is KylinOS V10. The Brotli implementation is sourced from its Python module (version 1.1.0). The 7z compressor is version 24.09, the zpaq compressor is version 7.15, and the zstd compressor is version 1.5.6.

Parameters The parameters are set to provide the best compression rates for each lossless audio codec, including the baselines and our proposed method. Table 2 summarizes the parameter configurations of the baselines.

Baselines The baselines include several traditional lossless audio compression methods and a large-model-based approach. The traditional baselines include Zstandard[12], Brotli[13], 7z[14], and zpaq[15]. Zstandard, or zstd, *is a fast lossless compression algorithm, targeting real-time compression scenarios at zlib-level and better compression rate*[12]. Brotli, developed by Google, is a hybrid compression algorithm that combines dictionary-based methods, entropy coding, and context modeling to achieve an effective balance between compression rate and decoding speed, particularly in web-related applications. Similarly, 7z employs the LZMA algorithm, a dictionary-based technique known for high compression ratios, albeit with high computational complexity. zpaq utilizes advanced context mixing techniques, achieving superior compression efficiency at the expense of speed. The large-model-based baseline is adapted from the work of a DeepMind team [11], and can be seen as an ablation of our LMCompress.

Program	Parameters
zstd	`--ultra -22`
Brotli	`brotli.compress(infile.read(), quality=11)`
7z	`a -mx9`
zpaq	`a -method 5`

Table 2 Configurations for the text compression baselines.

Potential Speedup The zstd, 7z, and zpaq implementations used in our experiments support explicit multi-threading options, but we set the number of threads to 1. The batch sizes of LMCompress and LMCompress* are both set to 1. This setting is for comparison fairness.

Fine-tuning The model we used for domain-aware text compression involves supervised Fine-Tuning on the base model. We leveraged LoRA[16] with hyperparameters $\alpha = 32$ and rank=8. We set the training batch size as 1 due to GPU Memory limitation, nevertheless, accumulates gradient every 8 steps. The model was trained for 10 epochs with a initial learning rate of 10^{-4}, while the first 10% of the steps were warm up steps. The base model was loaded in 8 bits precision and trained under bfloat16 precision.

S2. Detailed experimental results

While the compression rate is a key metric for evaluating compressors, time cost is also an important factor. This section provides detailed results on both compression rates and time costs for the compressors, including LMCompress and the baselines. Note that only encoding time is taken into account. Presently, we are not concerned about decoding time, as encoding time is more critical in many scenarios.

Throughout this section, "CR" in the tables refers to the compression rate. The arithmetic coding implementation we used is from [11].

Table 3 Image compression rates and time costs. The datasets are CLIC2019 and ILSVRC. As to the DeepMind method, the compression rates on ILSVRC are from [11], but the other metrics are left blank because neither such results nor Chinchilla models are publicly available.

Method	CLIC2019		ILSVRC	
	CR(%)	Time(s)	CR(%)	Time(s)
DeepMind-7B [11]			54.54	
DeepMind-70B [11]			48.07	
PNG [1]	47.08	38.46	54.63	35.56
JPEG-XL [3]	31.16	96.48	39.03	85.16
WebP [2]	35.56	30.98	42.37	38.93
JPEG-2000 [4]	48.64	16.25	54.42	22.08
LMCompress-bgpt	28.57	1196	34.01	1205
LMCompress-igpt	**14.62**	12.86 hours	**17.7**	21 hours

S2.1 Image

The results in Table 3 correspond to the experiments in main text Fig. 3 , Section 2.1 , except that a new method LMCompress-bgpt is included. LMCompress-bgpt is adapted from LMCompress by replacing iGPT with bGPT-image, a version of bGPT trained on raw image bytes.

We make two observations. First, regarding the compression rate, LMCompress nearly halves that of LMCompress-bgpt, while LMCompress-bgpt outperforms all other baselines. Second, LMCompress-bgpt consumes an order of magnitude less time than LMCompress, though its time cost remains higher than that of other baselines. This may be attributed to the relatively small size of bGPT-image, which has only 0.3 billion parameters.

Therefore, LMCompress-bgpt serves as a good alternative to LMCompress for balancing compression rate and time cost. More importantly, we can anticipate that emerging techniques for large models, such as size reduction and acceleration, will enable LMCompress to be fast enough meanwhile maintaining a low compression rate.

S2.2 Video

The results in Table 4 correspond to the experiments in Fig. 4 Section 2.2.

S2.3 Audio

The results in Table 5 correspond to the experiments in Fig. 5, Section 2.3 .

We observe that LMCompress achieves the best compression rates while consuming less time than most traditional baselines. This may be due to the underlying large model, bGPT-audio, which is designed to be audio-friendly and has only 0.3 billion parameters. If only encoding time and compression rate are considered, LMCompress is ready for practical audio compression.

Table 4 Video compression rates and time costs. The datasets include low-resolution video clips and high-resolution video clip.

Method	low-resolution(static scene)		low-resolution(dynamic scene)		high-resolution	
	CR(%)	Time(s)	CR(%)	Time(s)	CR(%)	Time(s)
zstd [12]	50.47	94.53	61.83	100.49	51.04	615.37
FFV1 [7]	38.02	3.3	46.94	3.7	40.16	4.16
H.264 [5]	29.58	3.6	39.52	3.8	37.42	7.36
H.265 [6]	21.05	15.8	28.49	14.45	21.69	29.91
LMCompress-igpt	**13.05**	30.85 hours	**19.61**	36.03 hours	**15.31**	144.47 hours

Table 5 Audio compression rates and time costs. The datasets include LJSpeech, Mozilla Common Voice 11, and VoxPopuli.

Method	LJSpeech		Mozilla Common Voice 11		VoxPopuli	
	CR(%)	Time(s)	CR(%)	Time(s)	CR(%)	Time(s)
FLAC [8]	41.26	31.42	35.53	27.12	39.99	25.99
OptimFROG [10]	25.42	531.90	19.15	511.53	27.19	540.47
WavPack [9]	33.15	288.76	25.55	296.65	34.21	231.25
DeepMind [11]	24.64	3.29 hours	19.63	2.89 hours	26.30	3.34 hours
LMCompress-bgpt	**18.26**	273.81	**12.31**	274.18	**20.81**	278.57

Table 6 Text compression rates and time costs. The datasets are MeDAL and Pile of Law. The LMCompress* method is adapted from [11, 17] by replacing LLaMA2-7B with LLaMA3-8B for a fair comparison. It is similar to LMCompress but utilizes the raw LLaMA3-8B model without fine-tuning.

Method	MeDAL		Pile of Law	
	CR(%)	Time(s)	CR(%)	Time(s)
zstd [12]	30.45	71.52	20.13	105.35
7z [14]	30.25	50.46	19.42	56.67
zpaq [15]	25.11	273.12	15.94	352.98
brotli [13]	27.93	171.01	18.02	221.78
LMCompress* [11, 17]	9.08	2744.77	5.86	5374.92
LMCompress	**9.06**	3071.86	**4.90**	6024.44

S2.4 Text

The results in Table 6 correspond to the experiments in Fig. 6, Section 2.4.

S3. Effects of Context Length

The context length is a critical hyperparameter that significantly influences the performance of both large models and traditional compression methods. In this section, we

conduct an empirical evaluation to analyze the impact of context length on LMCompress and traditional methods. For each data type, we select one representative dataset and evaluate LMCompress using three different context lengths, while traditional methods are tested with four context lengths. In the case of traditional methods, the symbol '∞' denotes the scenario where all related data is merged into a single file before compression. We report the compression rate, encoding time, peak GPU memory usage, and peak RAM consumption for LMCompress across different context lengths, whereas for traditional methods, we record only the compression rate and encoding time. The results are shown in Tables 7, 8, 9, 10 , and 11. We have three observations.

First, the length of context window have negligible effects on Peak memory usage for LMCompress.

Second, the context length has a strong impact on encoding time. Specifically, a longer context results in lower encoding time. This may be because a longer context reduces the number of large model invocations, thereby incurring less overhead and the same for the traditional.

Third, the context length also affects the compression rate. A longer context generally leads to a lower compression rate, possibly because it allows the large model to generate more accurate next-token probabilities. Similarly, traditional compression methods also rely on contextual information during compression, exhibiting the same trend where longer contexts improve compression performance. However, this effect weakens and becomes negligible when the context length reaches a sufficiently large value.

Table 7 The effects of context length on image compression using LMCompress. The dataset is CLIC2019.

Context Length	Compression Rate	Compression Time	Peak GPU Memory Usage	Peak RAM Usage
1024	14.62%	12.86h	5.63 GB	1.34 GB
768	15.38%	16.43h	5.63 GB	1.34 GB
512	17.54%	24.85h	5.63 GB	1.34 GB

Table 8 The effects of the context length on audio compression using LMCompress. The dataset is VoxPopuli.

Context Length	Compression Rate	Compression Time	Peak GPU Memory Usage	Peak RAM Usage
8192	20.81%	278.57s	0.23 GB	3.84 GB
6144	20.43%	331.87s	0.23 GB	3.93 GB
4096	20.66%	450.13s	0.23 GB	3.93 GB

Table 9 The effects of the context length on text compression using LMCompress. The dataset is MeDAL.

Context Length	Compression Rate	Compression Time	Peak GPU Memory Usage	Peak RAM Usage
8192	9.06%	3071.86s	15.49 GB	1.85 GB
6144	9.16%	6855.18s	15.49 GB	1.90 GB
4096	9.36%	7323.03s	15.49 GB	1.89 GB

Table 10 The effects of the context length on audio compression using traditional methods. The dataset is VoxPopuli.

Method	8192		16,384		32,768		∞	
	CR(%)	Time(s)	CR(%)	Time(s)	CR(%)	Time(s)	CR(%)	Time(s)
OptimFROG	29.00	581.38	28.16	566.68	27.69	561.05	27.13	531.44
WavPack	35.22	2227.89	34.60	1166.98	34.38	636.65	34.18	134.83

Table 11 The effects of the context length on text compression using traditional methods. The dataset is MeDAL.

Method	8192		16,384		32,768		∞	
	CR(%)	Time(s)	CR(%)	Time(s)	CR(%)	Time(s)	CR(%)	Time(s)
7z	42.02	53.49	38.21	25.84	35.26	13.68	23.89	5.57
zstd	39.54	45.11	37.05	23.48	34.83	11.92	24.05	5.64
brotli	31.51	16.77	30.50	15.69	29.74	16.04	24.29	0.5
zpaq	47.83	33.30	38.14	17.28	32.22	8.38	18.94	0.04

S4. Arithmetic coding

Our approach uses arithmetic coding [18] for data compression, so we introduce it in more detail to make the paper self-contained.

S4.1 A brief introduction to arithmetic coding

Arithmetic coding aims to map a sequence over a finite alphabet Σ into a value in the interval $[0, 1)$. Each sequence $\omega = \omega_1\omega_2\cdots\omega_n \in \Sigma^n$ is associated with a probability mass function p_n that satisfies $p_n(\omega) = \sum_{x\in\Sigma} p_{n+1}(\omega x)$. The arithmetic encoder compresses the sequence symbol by symbol while iteratively narrowing down the interval $I_0 = [0, 1)$.

Specifically, for any $1 \le k \le n$, suppose $I_{k-1} = [a_{k-1}, b_{k-1})$ is the interval obtained after encoding ω_{k-1}. To encode ω_k, we define the interval $I_k = [a_k, b_k)$ as follows:

$$a_k = a_{k-1} + (b_{k-1} - a_{k-1}) \sum_{x\in\Sigma, x\prec\omega_k} p(x|\omega_{1:(k-1)}), \tag{1}$$

$$b_k = a_{k-1} + (b_{k-1} - a_{k-1}) \sum_{x\in\Sigma, x\preceq\omega_k} p(x|\omega_{1:(k-1)}), \tag{2}$$

where

$$p(x|\omega_{1:(k-1)}) = \frac{p_k(\omega_{1:(k-1)}x)}{p_{k-1}(\omega_{1:(k-1)})}$$

Finally, we select a value $\lambda \in I_n$ that has the shortest binary representation and use it as the arithmetic code for ω. Note that the length $l(\lambda)$ of the binary representation of λ is

$$\begin{aligned} l(\lambda) &\leq \lceil -\log_2(b_n - a_n) \rceil + 1 \\ &= \left\lceil -\log_2 \prod_{i=1}^{n} p(\omega_i|\omega_{1:(k-1)}) \right\rceil + 1 \\ &= \lceil -\log_2 p_n(\omega) \rceil + 1. \end{aligned}$$

On the other hand, given λ, we can reconstruct the sequence ω in a similar manner. Starting with the interval $I_0 = [0, 1)$, assume that we have recovered $\omega_{1:(k-1)}$ and obtained the interval $I_{k-1} = [a_{k-1}, b_{k-1})$ that contains λ. The kth element of ω is the unique symbol $\omega_k \in \Sigma$ for which $I_k = [a_k, b_k)$ contains λ, where a_k and b_k are defined as shown in Formulas (1) and (2). This way, the original sequence ω can be exactly reconstructed from its arithmetic code λ, thus making arithmetic coding a lossless compression method.

S4.2 Arithmetic coding with unknown posterior distribution

A common scenario in practice is that the posterior distribution p is not known. In order to apply arithmetic coding, we have to find an approximate distribution q that is close to p. This is a long-time challenge dating back to 1960's when the well-known Solomonoff induction was proposed [19].

Solomonoff induction approximates the ground-truth distribution q with theoretical guarantee, but it relies on on the uncomputable semimeasure Solomonoff prior [20]. Recent years has witnessed many generative large models, for example, GPT series, LLaMA, and Chinchilla, which essentially estimate the ground truth distribution p from a huge corpus. It is known [21, 22] that, if fully trained with log-loss function, the estimated distribution q converges to the ground truth probability p. Hence, we can bypass uncomputability of Solomonoff prior via large models.

In arithmetic coding, we use q instead of p and hence need $\lceil -\log_2 q_n(\omega) \rceil + 1$ bits to represent a sequence $\omega \in \Sigma^n$. Therefore, the expected number of bits needed to compress a sequence in Σ^n is

$$\begin{aligned} &\mathbb{E}_{\omega \sim p_n} \lceil -\log_2 q_n(\omega) \rceil + 1 \\ \leq &\mathbb{E}_{\omega \sim p_n} [-\log_2 q_n(\omega)] + 2 \\ = &H(p_n, q_n) + 2. \end{aligned}$$

Here, $H(p_n, q_n)$ is the cross-entropy between the true distribution and the estimated distribution. According to the theory of cross-entropy, the better p_n is approximated, the better the compression is, and the shortest possible length of the arithmetic code is 2 plus the entropy of p_n.

References

[1] Boutell, T.: RFC2083: PNG (Portable Network Graphics) Specification Version 1.0. https://www.w3.org/TR/REC-png-961001

[2] Ooms, J.: Webp: A New Format for Lossless and Lossy Image Compression. R package version 1.2.1. https://github.com/jeroen/webp

[3] Alakuijala, J., Asseldonk, R., Boukortt, S., Bruse, M., Comșa, I.-M., Firsching, M., Fischbacher, T., Kliuchnikov, E., Gomez, S., Obryk, R., Potempa, K., Rhatushnyak, A., Sneyers, J., Szabadka, Z., Vandevenne, L., Versari, L., Wassenberg, J.: JPEG XL next-generation image compression architecture and coding tools. In: Tescher, A.G., Ebrahimi, T. (eds.) Applications of Digital Image Processing XLII, vol. 11137, p. 111370. SPIE, San Francisco, California, United States (2019). https://doi.org/10.1117/12.2529237 . International Society for Optics and Photonics

[4] Skodras, A., Christopoulos, C., Ebrahimi, T.: The jpeg 2000 still image compression standard. IEEE Signal Processing Magazine **18**(5), 36–58 (2001) https://doi.org/10.1109/79.952804

[5] Richardson, I.E.: The H.264 Advanced Video Compression Standard, 2nd edn., p. 368. Wiley Publishing, Chichester, UK (2010). https://doi.org/10.1002/9780470516935

[6] International Telecommunication Union (ITU): High Efficiency Video Coding (HEVC) - ITU-T Recommendation H.265. https://www.itu.int/rec/T-REC-H.265

[7] Niedermayer, M., Rice, D., Martinez, J.: FFV1 Video Coding Format Version 4. Internet-Draft draft-ietf-cellar-ffv1-v4-22, Internet Engineering Task Force (January 2024). Work in Progress. https://datatracker.ietf.org/doc/draft-ietf-cellar-ffv1-v4/22/

[8] Foundation, X.O.: FLAC: Free Lossless Audio Codec. https://xiph.org/flac/ (2023)

[9] Bryant, D.: Hybrid Lossless Wavefile Compressor. https://www.wavpack.com/index.html (2024)

[10] Ghido, F.: OptimFROG. http://losslessaudio.org/ (2022)

[11] Deletang, G., Ruoss, A., Duquenne, P.-A., Catt, E., Genewein, T., Mattern, C., Grau-Moya, J., Wenliang, L.K., Aitchison, M., Orseau, L., Hutter, M., Veness, J.: Language modeling is compression. In: The Twelfth International Conference on Learning Representations (2024)

[12] Meta Platforms, I.: Zstandard - Real-time data compression algorithm. https://facebook.github.io/zstd/ (2024)

[13] Alakuijala, J., Farruggia, A., Ferragina, P., Kliuchnikov, E., Obryk, R., Szabadka, Z., Vandevenne, L.: Brotli: A general-purpose data compressor. ACM Transactions on Information Systems (2019)

[14] Pavlov, I.: 7-Zip. https://www.7-zip.org/ (2024)

[15] Mahoney, M.: ZPAQ. https://mattmahoney.net/dc/zpaq.html (2016)

[16] Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: Lora: Low-rank adaptation of large language models. In: ICLR 2022 (2022)

[17] Huang, C., Xie, Y., Jiang, Z., Lin, J., Li, M.: Approximating human-like few-shot learning with gpt-based compression. arXiv preprint arXiv:2308.06942 (2023)

[18] Rissanen, J.J.: Generalized kraft inequality and arithmetic coding. IBM Journal of research and development **20**(3), 198–203 (1976)

[19] Solomonoff, R.: A formal theory of inductive inference. Inform. control **7**(1), 1–22 (1964)

[20] Li, M., Vitányi, P.: An Introduction to Kolmogorov Complexity and Its Applications. Springer, New York (2019). https://doi.org/10.1007/978-3-030-11298-1

[21] Ortega, P.A., Wang, J.X., Rowland, M., Genewein, T., Kurth-Nelson, Z., Pascanu, R., Heess, N.M.O., Veness, J., Pritzel, A., Sprechmann, P., Jayakumar, S.M., McGrath, T., Miller, K.J., Azar, M.G., Osband, I., Rabinowitz, N.C., György, A., Chiappa, S., Osindero, S., Teh, Y.W., Hasselt, H.V., Freitas, N., Botvinick, M.M., Legg, S.: Meta-learning of sequential strategies. ArXiv **abs/1905.03030** (2019)

[22] Grau-Moya, J., Genewein, T., Hutter, M., Orseau, L., Deletang, G., Catt, E., Ruoss, A., Wenliang, L.K., Mattern, C., Aitchison, M., Veness, J.: Learning universal predictors. In: Proceedings of the 41st International Conference on Machine Learning, pp. 16178–16205 (2024)