Your Dataset is a Multiset and You Should Compress it Like One

Daniel Severo* [1 2 3] **James Townsend*** [4] **Ashish Khisti**[2] **Alireza Makhzani**[3 2]

Karen Ullrich[1]

[1]Facebook AI Research
`dsevero@fb.com`
`karenu@fb.com`

[2]University of Toronto
`d.severo@mail.utoronto.ca`
`akhisti@ece.utoronto.ca`

[3]Vector Institute for AI
`makhzani@vectorinstitute.ai`

[4]University College London
`james.townsend@cs.ucl.ac.uk`

Abstract

Neural Compressors (NCs) are codecs that leverage neural networks and entropy coding to achieve competitive compression performance for images, audio, and other data types. These compressors exploit parallel hardware, and are particularly well suited to compressing i.i.d. batches of data. The average number of bits needed to represent each example is at least the well-known cross-entropy. However, the cross-entropy bound assumes the order of the compressed examples in a batch is preserved, which in many applications is not necessary. The number of bits used to implicitly store the order information is the logarithm of the number of unique permutations of the dataset. In this work, we present a method that reduces the bitrate of any codec by exactly the number of bits needed to store the order, at the expense of shuffling the dataset in the process. Conceptually, our method applies bits-back coding to a latent variable model with observed symbol counts (i.e. multiset) and a latent permutation defining the ordering, and does not require retraining any models. We present experiments with both lossy off-the-shelf codecs (WebP) as well as lossless NCs. On Binarized MNIST, lossless NCs achieved savings of up to 7.6%, while adding only 10% extra compute time.

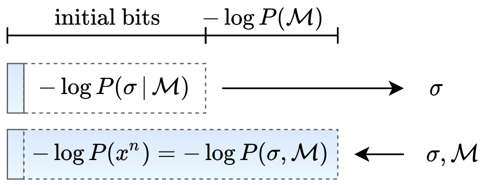
1 Introduction

A data source is usually modeled as an *ordered sequence* of random variables with some joint distribution. The objective of lossless data compression is to map each trajectory, i.e. the sequence of instances, to a compact representation such as a string of bits. The average number of bits of the smallest lossless representation for any source is completely characterized by it's probability distribution, and is known as the *entropy*. Compression algorithms such as Arithmetic Coding (AC) and Asymmetric Numeral Systems (ANS) [3] can compress any sequence to a size very close to the entropy, if the probability distribution is known.

In real-world applications, the source distribution is not known and must be estimated from data. The better the estimate, the closer the size of the representation is to the entropy. Deep generative models are good estimators for this task, and have recently been shown to reach competitive compression performance when paired with AC and ANS [7, 15, 17, 8, 1, 2, 6]. The pairings of these models with compression algorithms are increasingly being referred to as *Neural Compressors* (NCs) [9].

*Equal contribution.

In the compression literature, the elements of an ordered sequence are referred to as *symbols*. AC and ANS preserve the ordering of symbols in the input sequence. However, there are data types where order is not meaningful, such as collections of files, rows in a database, nodes in a graph, and, notably, datasets in machine learning applications. Formally, these are known as *multisets*: a generalization of a set that allows for repetition of elements. A multiset can be compressed with NCs by somehow ordering the elements and communicating the corresponding ordered sequence, but this is wasteful, because the order contains information, and therefore more bits are used to represent the source than are truly necessary.



initial bits
$-\log P(\mathcal{M})$
$-\log P(\sigma \mid \mathcal{M})$
σ
$-\log P(x^n) = -\log P(\sigma, \mathcal{M})$
$\sigma, \mathcal{M}$

Figure 1: Our entropy coder for datasets (multisets) is equivalent to bits-back coding on a latent variable model with observed multisets and latent permutations.

Current entropy coders tailored to multisets of i.i.d. symbols either have a sub-optimal compression rate or compute time which scales linearly with data dimensionality, making them unsuitable for multisets of images and other high dimensional sources [13, 14, 12]. To leverage the efficiency and compression performance of existing NCs, while simultaneously compressing a multiset optimally, requires some mechanism that can *forget the ordering* between symbols during encoding. In this work, we present such a method based on bits-back coding, and the key observation that any model over sequences can be seen as a latent variable model, where the observed variable is the unordered multiset, and the latent variable is a permutation that defines the order between symbols [18]. A more general, detailed exposition of the core multiset compression method is given in [11].

2 Background

Given access to some model P over sequences $x^n = (x_1, \dots, x_n)$ of discrete elements, entropy coders are functions that map x^n to bit-strings of approximately $-\log P(x^n)$ in length. Under the i.i.d. assumption $x_i \sim P_d$, the average bit-length $-\frac{1}{n}\log P(x^n) = -\frac{1}{n}\sum_{i=1}^{n}\log P(x_i)$ approaches the *sequence-cross-entropy* $H(P_d) + D_{\mathrm{KL}}(P_d \,\|\, P)$ for large n, where $H(P_d)$ is the entropy of the data source. Sequence-cross-entropy is the smallest possible average bit-length that can be achieved for a given model P. We say an entropy coder is *optimal* if it can compress sequences close to the sequence-cross-entropy.

The entropy coder we develop in this paper depends on another well known entropy coder called “asymmetric numeral systems” (ANS) [3]. In contrast to Arithmetic Coding, ANS is last-in-first-out (i.e. stack-like), as the encoded sequence x^n is decoded in reverse order $x_n, \dots, x_1$. ANS encodes sequences into a single, large, natural number s. The outputted bit-string is just the binary representation of s, which is $\log s$ bits in size. Each encoded symbol x_i increases the bit-string by approximately $-\log P(x_i)$, and decreases it by the same amount during decoding.

A key property of ANS is that performing a decode operation using distribution P will produce samples distributed according to P. Decoding reduces the ANS state, which can be viewed as a *random seed* that is slowly consumed as samples are drawn. The random seed can be recovered by performing an encode with the sampled symbol. Therefore, ANS can be used as an *invertible sampler*. Invertible sampling is also possible with other entropy coders and is commonly known as ‘bits-back coding’ [4]. However, the stack-like nature of ANS allows for interleaving of sampling (decoding) and encoding with different distributions. This observation was first made by [15], and used to compress images with *latent variable models*. Follow-up works propose elaborations on the original idea for specific classes of latent variable models [8, 17, 16].

3 Method

In this section we present a method that converts an entropy coder for sequences x^n of i.i.d. symbols into one for multisets $\mathcal{M}$ of i.i.d symbols. We describe only the compression scheme, as decompression follows from all steps being invertible. The multiset-cross-entropy achieved by compressing $\mathcal{M}$ is at most $\frac{1}{n}\log n! \approx \log n$ bits less than the usual sequence-cross-entropy resulting from compressing

x^n, with equality when there are no repeated symbols. This is exactly the information required to represent the ordering. Our method is optimal in the sense that it can recover all $\frac{1}{n}\log n!$ bits, irrespective of how well the original sequence codec performs on x^n. The naive implementation discussed next can be seen as a McBits [10] entropy coder with an exact posterior, where the joint distribution is the entropy coder chosen for conversion.

Key to our method is that any model over sequences can be seen as a latent variable model over observed multisets (i.e. symbol counts) and latent permutations defining the ordering [18]. To see how, let $[x^n]$ represent the set of all possible permutations of x^n. All sequences in this set have the same frequency count of symbols, and therefore $[x^n]$ has a one-to-one correspondence with a unique multiset $\mathcal{M}$. The amount of permutations $|[x^n]| \leq n!$ is equal to the *multinomial coefficient* calculated from symbol counts in $\mathcal{M}$, with equality when there are no repeated symbols. If the symbols x_i are i.i.d. under model P, then all sequences in $[x^n]$ have mass $P(x^n)$, implying

$$P(\mathcal{M}) = P([x^n]) = |[x^n]|P(x^n). \tag{1}$$

A source that generates sequences x^n of i.i.d. symbols can thus be seen as a compound source that first selects a multiset $\mathcal{M}$ and then picks an order uniformly at random from the corresponding set $[x^n]$. If σ represents a positive integer that indexes into $|[x^n]|$, then the joint and posterior distributions under model P are

$$P(\sigma \,|\, \mathcal{M}) = \frac{1}{|[x^n]|}, \quad P(\mathcal{M}, \sigma) = P(x^n) = \prod_{i=1}^{n} P(x_i). \tag{2}$$

The distribution $P(\mathcal{M})$ does not factorize over symbols, making it inefficient for entropy coding. For example, we can compress $\mathcal{M}$ by always picking $\tilde{x}^n \in |[x^n]|$ such that $\tilde{x}_1 \leq \tilde{x}_2 \leq \cdots \leq \tilde{x}_n$, together with $P(\tilde{x}_i \,|\, \tilde{x}^{i-1})$. This requires updating the mass for all symbols $a \geq \tilde{x}_i$ in the alphabet at each step i. In contrast, $P(x^n)$ factorizes over x_i and hence does not require updates, but uses exactly $\log |[x^n]|$ surplus bits. By using invertible sampling to randomly pick *any* sequence in $|[x^n]|$, we can compress efficiently with $P(x^n)$ while removing exactly the surplus bits from the ANS state. Equivalently, we augment the multiset to a sequence by pairing it with an order [10], and then recover the bits using bits-back (Figure 1). First, an ANS state s is initialized and invertible sampling (decoding) is performed with $P(\sigma \,|\, \mathcal{M})$ to sample an index σ. The sequence $\tilde{x}^n \in [x^n]$ corresponding to σ is then encoded with $P(\tilde{x}^n) = P(\mathcal{M}, \sigma)$. Sampling reduces s, while encoding increases it. Overall, the bit-length of s increases by exactly the number of bits required to represent $\mathcal{M}$

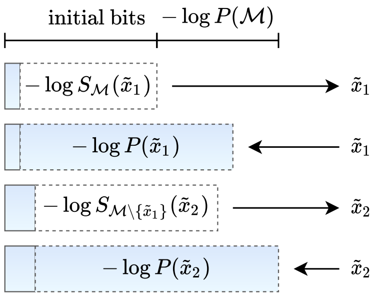


Figure 2: Amortizing over symbols allows us to compress a single multiset $\mathcal{M}$ while still achieving the information content $-\log P(\mathcal{M})$.

$$\log P(\sigma \,|\, \mathcal{M}) - \log P(\mathcal{M}, \sigma) = -\log P(\mathcal{M}). \tag{3}$$

The scheme as described above would only be optimal when amortizing over a collection of multisets, due to the $\log s$ *initial bits* required to initialize the ANS state s. It is possible to compress a single multiset while still achieving $-\log P(\mathcal{M})$ via an incremental version of our method that amortizes over symbols. At each step, a symbol is *sampled without replacement* from the multiset and immediately encoded with P. Sampling is performed using ANS with distribution $S_{\mathcal{M}}$, which assigns probabilities proportional to the frequency counts in $\mathcal{M}$. This is shown in Figure 2 for a multiset of size 2. The sequence $\tilde{x}^n$ generated from the invertible sampling steps (see Section 2) is a random permutation of x^n. Therefore, the order information between symbols in x^n has been destroyed, as only $\tilde{x}^n$ is stored. This implies our method is optimal, since it removes the maximum amount of redundant order information ($\log |[x^n]|$), and achieves the multiset-cross-entropy if P achieves the sequence-cross-entropy.

Sampling without replacement from the multiset can be done efficiently by using a *binary search tree* (BST), which we describe in detail in [11]. Traversing the BST requires comparing symbols under a fixed (usually lexicographic) ordering. In the extreme case where symbols in some alphabet $\mathcal{A}$ are represented by binary strings of length $\log |\mathcal{A}|$, a single comparison between symbols would require

$\log|\mathcal{A}|$ bit-wise operations. However, 'short-circuit' evaluations, where the next bits are compared only if all previous comparisons result in equality, make this exponentially unlikely. All operations required have worst-case and average complexity equivalent to that of a search on a balanced BST. Overall, our method adds $\mathcal{O}(n \log m)$ in average compute time to the original sequence codec, where n and m are the total and unique number of symbols in the multiset.

4 Experiments

In the experiments that follow, we used the ANS implementation in the Craystack library [17]. Details of each experiment are discussed further in the appendix.

Toy multisets We compressed synthetically generated toy multisets to provide evidence of the computational complexity and optimal compression rate of the method. The alphabet $\mathcal{A}$ is always a subset of $\mathbb{N}$. For each run, we generate a multiset with $m = 512$ unique and $|\mathcal{M}| = n$ total symbols. The true data distribution is used for P, which is sampled from a Dirichlet prior with coefficients $\alpha_k = k$ for $k = 1, \ldots, |\mathcal{A}|$. Results are shown in Section 4 averaged over 20 runs, with shaded regions representing the 99% to 1% confidence intervals. The new codec compresses a multiset close to it's information content ($-\log P(\mathcal{M})$) for varying alphabet sizes, as can be seen in the top plot. The total encode plus decode time is unaffected by the alphabet size $|\mathcal{A}|$. The total time scales linearly with the multiset size $|\mathcal{M}|$ (bottom plot), as expected.

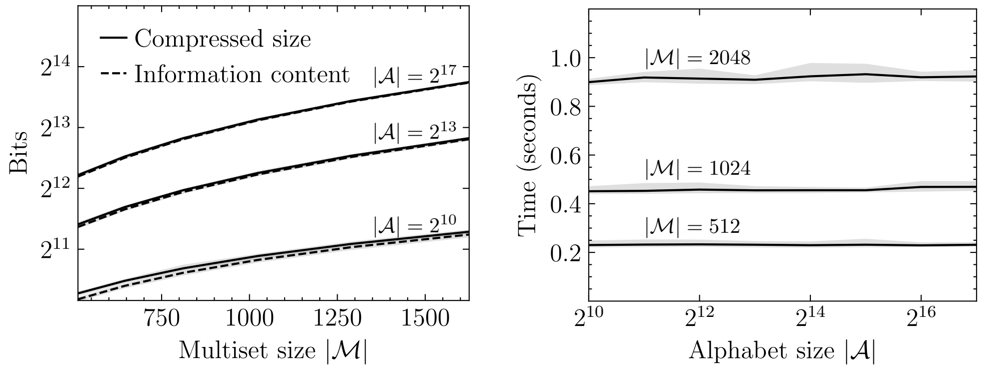


Figure 3: Left: Final compressed length is close to the information content $-\log P(\mathcal{M})$ for varying alphabet and multiset sizes. Right: Computational complexity does not scale with alphabet size $|\mathcal{A}|$, and is linear in $|\mathcal{M}| = n$.

Lossless Neural Compression on Binarized MNIST Here, a lossless compression algorithm called 'Bits-back with ANS' (BB-ANS) [15] was used to compressed Binarized MNIST. BB-ANS is an entropy coder for latent variable models with factorized joint $P(x, z) = P(z)P(x \mid z)$, and approximate posterior $Q(z \mid x)$, over observed x and latent z. The observed model $P(x \mid z)$ is composed of a factorized Bernoulli distribution over each pixel, while the approximate posterior Q is a factorized Gaussian. The average bit-length achieved by BB-ANS is an upper-bound on the sequence-cross-entropy. It is equal to the *Negative Evidence Lower Bound* of the discretized model, with equality when the approximate posterior perfectly matches the true posterior $P(z \mid x)$. We used the pre-trained model and code made publicly available by the author[2]. First, invertible sampling is performed to select a binarized MNIST image for compression. Then, BB-ANS is applied as described in the experimental section of [17]. This process is repeated, until all $10,000$ images are compressed. We compared the average bit-length with and without the invertible sampling step. Results are shown in Table 1. Since the images are all unique, the maximum theoretical savings is $\log(10,000!)$ bits ≈ 14 kB. This represents a potential savings of 7.6%, which is achieved by our method, at the cost of 10% extra compute time on average.

[2]`https://github.com/bits-back/bits-back`

Table 1: Savings incurred by our method on Binarized MNIST (BMNIST) with BB-ANS, and standard MNIST with lossy WebP. The maximum theoretical savings for both datasets is $\log(10,000!)$ bits $\approx$ 14 kB, which is reached by our method. The last column shows the added mean $\pm$ 3 std. compute time from using our method over 100 runs.

	Bits-per-pixel		Relative savings		
Dataset	Ordered	Unordered (ours)	Theoretical	Actual	Extra compute time
BMNIST	0.198	**0.183**	7.6%	**7.6%**	10% ± 4%
MNIST	1.031	**1.016**	1.5%	**1.5%**	37% ± 8%

Compressing standard MNIST with lossy WebP We showcase our method with WebP, an off-the-shelf lossy codec, on standard MNIST. WebP is a lossy compression algorithm that outputs a variable-length sequence of bytes which we call a *byte-array*. To encode, we first perform invertible sampling to select an image from the test set. WebP is applied to the image, and the bytes in the byte-array are encoded sequentially into the ANS state using a uniform distribution. Finally, because the byte-array is variable in length, the length itself must also be encoded. As in the BB-ANS experiment, all $10,000$ images are compressed. Results are shown in Table 1. The byte-arrays are all unique and therefore the maximum theoretical savings in raw bits is also $\log(10,000!)$ bits $\approx$ 14 kB. This represents a potential savings of 1.5%, which is achieved by our method, at the cost of 37% extra compute time on average.

5 Acknowledgements

In preparing this research, authors affiliated to the Vector Institute were funded, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute. We would like to thank Tim Vieira, whose blog post "Heaps for incremental computation" inspired the binary search tree data structure[3]. All plots were made using the Science Plots package [5].

References

[1] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. "End-to-end optimized image compression". In: *arXiv preprint arXiv:1611.01704* (2016).

[2] Johannes Ballé et al. *Variational image compression with a scale hyperprior*. 2018. arXiv: `1802.01436 [eess.IV]`.

[3] Jarek Duda. "Asymmetric Numeral Systems". In: *arXiv:0902.0271 [cs, math]* (2009). arXiv: `0902.0271 [cs, math]`.

[4] Brendan J. Frey. "Bayesian Networks for Pattern Classification, Data Compression, and Channel Coding". PhD thesis. University of Toronto, 1997.

[5] John D. Garrett and Hsin-Hsiang Peng. "garrettj403/SciencePlots". Version 1.0.7. In: (Feb. 2021). DOI: `10.5281/zenodo.4106649`. URL: `http://doi.org/10.5281/zenodo.4106649`.

[6] Jonathan Ho, Evan Lohn, and Pieter Abbeel. *Compression with Flows via Local Bits-Back Coding*. 2020. arXiv: `1905.08500 [cs.LG]`.

[7] Diederik P. Kingma et al. *Variational Diffusion Models*. 2021. arXiv: `2107.00630 [cs.LG]`.

[8] Friso H. Kingma, Pieter Abbeel, and Jonathan Ho. "Bit-Swap: Recursive Bits-Back Coding for Lossless Compression with Hierarchical Latent Variables". In: *International Conference on Machine Learning*. Oct. 2019.

[9] Matthew Muckley et al. *NeuralCompression*. `https://github.com/facebookresearch/NeuralCompression`. 2021.

[10] Yangjun Ruan et al. "Improving Lossless Compression Rates via Monte Carlo Bits-Back Coding". In: *International Conference on Machine Learning*. 2021.

[11] Daniel Severo et al. *Compressing Multisets with Large Alphabets*. 2021. arXiv: `2107.09202 [cs.IT]`.

[3] `https://timvieira.github.io/blog/post/2016/11/21/heaps-for-incremental-computation/`

[12] Christian Steinruecken. “Compressing Combinatorial Objects”. In: *2016 Data Compression Conference (DCC)*. IEEE. 2016, pp. 389–396.

[13] Christian Steinruecken. “Compressing Sets and Multisets of Sequences”. In: *IEEE Transactions on Information Theory* 61.3 (2015), pp. 1485–1490.

[14] Christian Steinruecken. “Lossless Data Compression”. PhD thesis. University of Cambridge, 2014.

[15] James Townsend, Thomas Bird, and David Barber. “Practical Lossless Compression with Latent Variables Using Bits Back Coding”. In: *International Conference on Learning Representations (ICLR)*. 2019.

[16] James Townsend and Iain Murray. “Lossless Compression with State Space Models Using Bits Back Coding”. In: *Neural Compression: From Information Theory to Applications – Workshop at ICLR*. 2021.

[17] James Townsend et al. “HiLLoC: Lossless Image Compression with Hierarchical Latent Variable Models”. In: *International Conference on Learning Representations (ICLR)*. 2020.

[18] L.R. Varshney and V.K. Goyal. “Toward a Source Coding Theory for Sets”. In: *2006 Data Compression Conference (DCC)*. IEEE. 2006, pp. 13–22.

6 Appendix

In this section we present experiments on synthetically generated multisets with known source distribution, multisets of grayscale images with lossy codecs (MNIST), and collections of JSON maps represented as a multiset of multisets. We used the ANS implementation in the Craystack library [17] for all experiments.

6.1 Synthetic multisets

Here, synthetically generated multisets are compressed to provide evidence of the computational complexity and optimal compression rate of the method. We grow the alphabet size $|\,\mathcal{A}\,|$ while sampling from the source in a way that guarantees a fixed number of unique symbols $m = 512$. The alphabet $\mathcal{A}$ is always a subset of $\mathbb{N}$.

For each run, we generate a multiset with $m = 512$ unique symbols, and use a skewed distribution, sampled from a Dirichlet prior with coefficients $\alpha_k = k$ for $k = 1, \ldots, |\,\mathcal{A}\,|$, as the true data distribution D.

The final compressed size of the multiset and the information content (i.e. Shannon lower bound) assuming the distribution D is used, are shown in Section 4 for different settings of $|\mathcal{A}|$, alongside the total encode plus decode time. Results are averaged over 20 runs, with shaded regions representing the 99% to 1% confidence intervals. In general, the new codec compresses a multiset close to it's information content for varying alphabet sizes, as can be seen in the left plot.

The total encode plus decode time is unaffected by the alphabet size $|\,\mathcal{A}\,|$. As discussed previously, the overall complexity depends on that of coding under the chosen codec used with distribution D. Here, the codec does include a logarithmic time binary search over $|\,\mathcal{A}\,|$, but this is implemented efficiently and the alphabet size can be seen to have little effect on overall time. The total time scales linearly with the multiset size $|\mathcal{M}|$ (right plot), as expected.

6.2 MNIST with lossy WebP

We implemented compression of a multiset $\mathcal{M}$ of grayscale images using the lossy WebP codec. We tested on the MNIST test set, which is composed of $|\mathcal{M}| = 10,000$ distinct grayscale images of handwritten digits, each 28×28 in size. To encode the multiset, we perform the sampling procedure to select an image to compress, as usual. The output of WebP is a prefix-free, variable-length sequence of bytes, which we encoded into the ANS state via a sequence of `encode` steps with a uniform distribution. It is also possible (and faster) to move the WebP output directly into the lower-order bits of the ANS state.

We compared the final compressed length with and without the sampling step. In other words, treating the dataset as a multiset and treating it as an ordered sequence. The savings achieved by using our method are shown in Figure 4. The theoretical limit shown in the left plot is $\log |\mathcal{M}|!$, while in the right plot this quantity is divided by the number of bits needed to compress the data sequentially.

Note that the maximum savings per symbol $-\frac{1}{|\mathcal{M}|} \log_2 |\mathcal{M}|! \approx \log_2 |\mathcal{M}|$ depends only on the size of multiset. Therefore, when the representation of the symbol requires a large number of bits, the percentage savings are marginal (roughly 1.5% for 10,000 images, in our case). To improve percentage savings (right plot), one could use a better symbol codec or an adaptive codec which doesn't treat the symbols as independent. However, as mentioned, the savings in raw bits (left plot) would remain the same, as it depends only on the multiset size $|\mathcal{M}|$.

6.3 Collection of JSON maps as nested multisets

The method can be nested to compress a multiset of multisets, by performing an additional sampling step that first chooses which inner multiset to compress. In this section we show results for a collection of JSON maps $\mathcal{M} = \{\mathcal{J}_1, \ldots, \mathcal{J}_{|\mathcal{M}|}\}$, where each map $\mathcal{J}_i = \{(k_1, v_1), \ldots, (k_{|\mathcal{J}_i|}, v_{|\mathcal{J}_i|})\}$ is itself a multiset of key-value pairs. To compress, a depth-first approach is taken. First, some $\mathcal{J} \in \mathcal{M}$ is sampled without replacement. Key-value pairs are then sampled from $\mathcal{J}$, also without replacement, and compressed to the ANS state until $\mathcal{J}$ is depleted. This procedure repeats, until the outer multiset

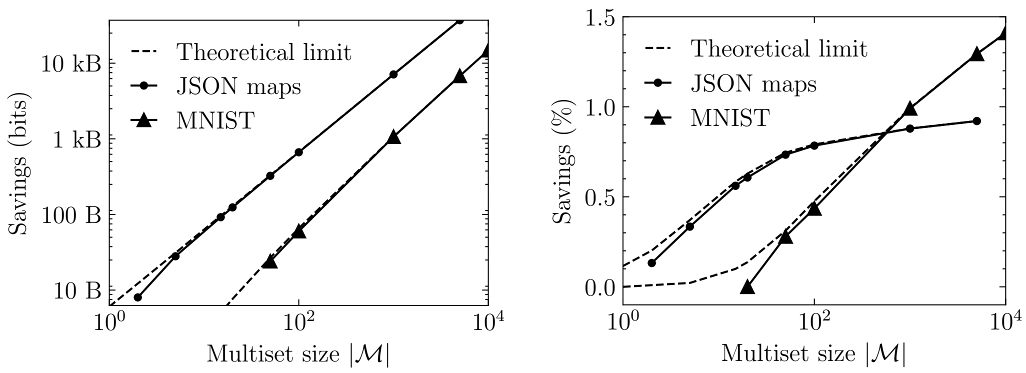


Figure 4: Left: Savings in raw bits. Right: Percentage savings. Rate savings due to using our method to compress a multiset instead of treating it as an ordered sequence. Savings are close to the theoretical limit in both MNIST and JSON experiments. The symbols are bytes outputted by lossy WebP for MNIST, and UTF-8 encoding for JSON maps. A uniform distribution over bytes is used to encode with ANS.

$\mathcal{M}$ is empty. Assuming all maps are unique, the maximum number of savable bits is

$$\log |\mathcal{M}|! + \sum_{i=1}^{|\mathcal{M}|} \log |\mathcal{J}_i|!. \tag{4}$$

The collection of JSON maps is composed of public GitHub user data taken from a release of the Zstandard project[4]. All key-value elements were cast to strings, for simplicity, and are encoded as UTF-8 bytes using a uniform distribtuion. Figure 4 shows the number and percentage of saved bits. The theoretical limit curve shows the maximum savable bits with nesting, i.e. eq. (4). Note that, without nesting, the theoretical limit would be that of MNIST (i.e. $\log |\mathcal{M}|!$). The method gets very close to the maximum possible savings, for various numbers of JSON maps. The rate savings were small, but these could be improved by using a better technique to encode the UTF-8 strings.

Assuming $\mathcal{J}$ represents the JSON map in $\mathcal{M}$ with the largest number of key-value pairs, and that the time complexity of comparing two JSON maps is $\mathcal{O}(|\mathcal{J}|)$, the complexity of the four BST operations for the *outer* multiset is $\mathcal{O}(|\mathcal{M}||\mathcal{J}| \log |\mathcal{M}|)$. The overall expected time complexity for both encoding and decoding is therefore $\mathcal{O}(|\mathcal{M}||\mathcal{J}|(\log |\mathcal{M}| + \log |\mathcal{J}|))$. We believe it may be possible to reduce this by performing the inner sampling steps in parallel, or by speeding up the JSON map comparisons.

[4]`https://github.com/facebook/zstd/releases/tag/v1.1.3`