Comparing Text Compression Capabilities of Large Language Models with Traditional Compression Algorithms

Mehran Haddadi and William John Teahan
School of Computer Science and Engineering, Bangor University, Bangor, Wales, UK
mhh24vrg@bangor.ac.uk and w.j.teahan@bangor.ac.uk

Abstract

This work evaluates the non-English and unstructured text compression performance of Large Language Models (LLMs) by comparing them with traditional baselines on datasets from eight most widely spoken languages. Experimental results show that the evaluated LLM (LLaMA-3.2-1B) was considerably outperformed by the baselines, particularly on non-English datasets, where its performance relative to the best baseline was more than three times worse than on English datasets on average. It also compressed unstructured English data up to more than twofold less effectively than plain English data. Traditional methods, however, remained largely dataset-agnostic. Surprisingly, the LLM achieved worse compression ratios on some datasets than others despite modeling them more accurately. Overall, the outcomes and substantially higher compression time and resource consumption indicate that current LLMs are highly impractical for the compression task, where traditional methods continue to excel. Codes are available at github.com/mehranhaddadi13/llm_compress.

1 Introduction

Given the large amount of data generated and transmitted by billions of the Internet users everyday, benefits of compression in storing and transmitting data is not obscure. Compression methods fall into lossy and lossless categories. The latter is suitable for text data since losing content after decompression spoils its integrity. They also are categorized into dynamic (online) and static (offline) compressors. The former adjusts its modeling based on the data being compressed while performing the compression, whereas the latter follows a pre-defined approach to model the data.

Since Shannon (1948) introduced entropy and redundancy in information theory , the theoretical lower bound of lossless compression of each data, numerous work has been done to approach this limit. Entropic compressors consist of probability modeling and coding parts; the latter transforms the probability distribution calculated by the former to an encoded representation. With coders performing near-optimally, such as arithmetic coding (Witten et al., 1987) and Huffman coding (Huffman, 1952), being commonly used, the variance in the performance of compressors is attributed to their modeling part.

While traditional algorithms rely on the statistics of the data, the number of unique tokens and their frequencies, to calculate probabilities, compressors with Neural Networks (NNs) as their modeling part utilize NNs to capture correlations hidden within data. Recently, triumphs of Large Language Models (LLMs) in language modeling have motivated researchers to utilize them in data compression. Experimental results have demonstrated dominant compression performance of LLMs, outperforming SOTA by margin. Given that the loss function of LLMs and the objective of the compression task are the same (cross entropy), Deletang et al. (2024) showed language modeling and compression are tightly connected. Almost all research has investigated the compression performance of LLMs on English data, e.g. Enwik 8 dataset (Mahoney, 2006), with other languages remaining highly under-represented. Considering that plain English text comprises the major part of their training data, LLMs supposedly do not compress other languages and unstructured data, such as log files, as well as plain English data. Therefore, recent datasets from the most widely spoken languages, namely English, Spanish, Chinese, Hindi, French, German, Arabic and Farsi, along with swapped and substituted versions of Enwik 8, are used to compare the compression performance of LLMs on English, non-English, and unstructured text against traditional compression methods.

2 Related Work

Early research on the compression performance of NNs was conducted on simple shallow networks. High-performance models outperforming previous approaches have emerged by the advent of more complex, yet effective, architectures, such as Recurrent Neural Networks (RNNs) (Elman, 1990), Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Transformers (Vaswani et al., 2017). Also, studies have been done on the performance of LLMs since their advent.

2.1 Compression Using Neural Networks

CIMIX (Knoll, 2014) uses a large number of models to obtain the probability of the next bit and uses LSTM to mix the context. Finally, the probability is generated via NNs and encoded using arithmetic encoding. It outperforms `lstm-compress` (Knoll, 2015), which only includes LSTMs, and `tensorflow-compress` (Knoll, 2017), which uses LSTMs in TensorFlow.

Cox (2016) and Goyal et al. (2018) utilized RNNs to calculate the probability distribution of the next token and then applied arithmetic encoding. TRACE (Mao et al., 2022b) uses a single-layer transformer to compress data. DecMac (Liu et al., 2019) utilizes LSTMs with arithmetic encoding. DZip (Goyal et al., 2021) integrates static and dynamic compression by mixing a bootstrap model and a supporter model. Bellard (2019) in NNCPv1 studied LSTM and Transformer architectures to calculate the probability of the next symbol to encode it by arithmetic encoding. NNCPv2 and NNCPv3 made improvements on NNCPv1's performance. OREO (Mao et al., 2022a) uses a batch-wise ordered mask on the input symbol history and a multi-layer perceptron to learn input features. PAC (Mao et al., 2023) uses two neural blocks to compress data.

MLMCompress (Öztürk and Mesut, 2024) encodes the index of the next word using a combination of NNs and Huffman encoding. MSDZip (Ma et al., 2025) utilizes a stepwise-parallel multi-GPU compression strategy and a mixing block to enhance compression speed. Heurtel-Depeiges et al. (2025) trained small vanilla Transformers on large data to perform compression. L3TC (Zhang et al., 2025) uses RWKA (Peng et al., 2023) to achieve compression ratios comparable to those of SOTA neural network compressors through using an outlier-aware tokenizer, while being significantly faster.

2.2 Compression Using Large Language Models

LLMZip (Valmeekam et al., 2023) uses LLaMA-7B (Touvron et al., 2023a) to calculate the sorted probability distribution of the next token and compress the ranking of the true token via a compression algorithm, e.g. arithmetic encoding. GPT-AC (Huang et al., 2023) integrates GPT and LLaMA2-7B (Touvron et al., 2023b) with arithmetic coding. Given a context, AlphaZip (Narashiman and Chandrachoodan, 2024) uses the output logits of different versions of GPT-2 to calculate the probability distribution of entire vocabulary tokens to obtain the true token's ranking, which is then compressed by GZip (Gailly) and Brotli (Alakuijala et al., 2018). Deletang et al. (2024) compared foundation models as compressors with standard compression algorithms on text, picture and audio modalities.

Gili Fernández De Romarategui (2024) used LLMZip's framework, but with lighter LLMs instead of LLaMA-7B for the sake of memory usage. Huang et al. (2024) showed the LLMs' performance on downstream tasks is linearly correlated with their compression performance on related corpora. FineZip (Mittu et al., 2024) significantly improved compression speed of LLMZip with a minor compression ratio drop by using *online memorization*, applying PEFT (Mangrulkar et al., 2022) on the input text, and dynamic context window. In LMCompress (Li et al., 2025), LLaMA3-8B (Grattafiori et al., 2024) is fine-tuned on domain-specific texts to enhance its performance.

Gilbert et al. (2023) prompted ChaptGPT-4 and ChatGPT-3.5 (OpenAI, 2022) to compress and decompress short English texts. Although they achieved high compression ratios, the results show they failed to compress losslessly. ALCZip (Wang and Zhang, 2024) utilizes GPT-2 to first simplify the input, followed by the creation of an adaptive dictionary of letter combinations, which is then used by Huffman encoding to generate the compressed output. It outperformed FineZip in compression speed.

Only AlphaZip studied compressing non-English languages by LLMs. Its results show GPT-2 compresses French not as well as English and expands Hindi instead of compressing. However, fine-tuning resulted in improvements.

3 Experimental Setup

All the experiments were conducted on the Hawk nodes of Supercomputing Wales (Supercomputer-Wales, 2025), equiped with Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, 200GB of memory, and Nvidia P100 GPUs with 16GB of GPU memory.

Using dynamic context length together with BZip2 instead of arithmetic coding enabled FineZip to achieve much faster, yet comparable, results to LLMZip. Hence, this work follows the implementation of FineZip to reduce computational costs. It calculates the probability distribution of a token given previous tokens using an LLM and organizes them in descending order to obtain rankings; i.e. if the true value of a token is at the top of the list of the predicted values, rank 0 is assigned to that token, if it is the second highest probable value, rank 1, and so on. These rankings are then compressed using BZip2, where higher rankings lead to better compression. It also uses the so-called "online memorization" step, which is fine-tuning the model on the data to be compressed using LoRA (Hu et al., 2021) to make it more probable for the model. Tawa toolkit (Teahan, 2018) provides the ability to train PPM models on the data to be compressed, making it a good baseline to compare the performance of the "online memorization" step of FineZip with. LLaMA-3.2-1B (Meta-AI, 2024) was chosen as the LLM since it is a lightweight multi-lingual model that fits in the 16GB GPU memory of an Nvidia P100 after 4-bit quantization. Table 1 provides a summary of the used models. The models' parameter settings are available in Appendix.

Model	Description
zlib	The Zlib module from Python standard library (Python Standard Library, c)
bz2	The BZip2 module from Python standard library (Python Standard Library, a)
lzma	The LZMA module from Python standard library (Python Standard Library, b)
pyppmd	The PyPPMd Python module from Hiroshi Miura (Miura, 2021)
Tawa	Dynamic Tawa toolkit
Tawa-ts	Static Tawa toolkit trained on the input
Tawa-td	Dynamic Tawa toolkit trained on the input
LLM	LLaMA-3.2-1B
LLM-FT	LLaMA-3.2-1B fine-tuned on the input

Table 1: Summary of the models

For each compression, training, and fine-tuning process, we measured the duration in seconds, the average GPU and CPU utilization in percentage, and the peak memory and GPU memory usage in megabytes (MB). Also, the compression ratio (the compressed size divided by the original size, where smaller values correspond to better compression performance) was reported. In addition, the percentage of the correctly predicted true tokens and the percentage of true tokens ranked between 0 and 15 by the LLM were calculated. The true tokens appearing among the top-15 most probable predictions are referred to "top-15" tokens. These two metrics indicate the accuracy of the LLM in modeling the input data.

Dataset	Source
enwik	Enwik 8 dataset
enwik-sub	Enwik 8 dataset with substituted characters
enwik-swap	Enwik 8 dataset with swapped words
book	100 MB of BookCorpus (Zhu et al., 2015)
en	100 MB of the Open Australian Legal Corpus (Butler, 2025)
ar	100 MB of Arabic Punctuation Dataset (Yagi and Elnagar, 2024)
ch	100 MB of Chinese Corpora Internet 3.0-HQ (Wang et al., 2024)
de	100 MB of the German part of Nemotron-CC dataset (MultiSynt, 2025; Su et al., 2025)
es	100 MB of esCorpius (Gutiérrez-Fandiño et al., 2022)
fa	100 MB of the Farsi part of LSCP dataset(Abdi Khojasteh et al., 2020)
fr	100 MB of the French part of French-English dataset (Bojar et al., 2015)
hindi	100 MB of Hindi-TinyStories (Singh, 2024)

Table 2: Summary of Datasets. Each dataset is 104,857,600 bytes.

Table 2 provides a summary of the used datasets. To obtain *enwik-sub*, each English character of *enwik* was substituted with another, resulting in a text file with meaningless words. For *enwik-swap*, words of *enwik* were swapped to make nonsense sentences from meaningful words. The Appendix includes detailed explanation of the creation of each dataset.

enwik, book and *en* are referred to as **English datasets** while the term **non-English datasets** represents seven datasets which are not in English.

4 Results and Analysis

In this section, first the compression results of the LLM on English and non-English datasets are compared with those of baselines. Second, the outcomes of compressing *enwik, enwik-sub* and *enwik-swap* by the LLM and traditional methods are juxtaposed.

4.1 Compressing English and non-English datasets

Table 3 shows although the LLM could surpass *zlib* and *bz2* on some English and a few non-English datasets, it was outweighed by *lzma, pyppmd* and all Tawa models in terms of achieved compression ratios. The slight performance deterioration of *LLM-FT* on *enwik* and *en* compared to *LLM* offset the 5% improvement observed on the *book* dataset, leading to an insignificant 0.4% fine-tuning gain on the English datasets on average. On the other hand, *LLM* witnessed significant improvement after fine-tuning on non-English datasets, especially *hindi* (24%), *fa* (13%), and *ar* (13%), resulting in 9% better compression ratio for *LLM-FT* on non-English data on average. Similarly, training *Tawa* improved its performance on all datasets. All the baselines compressed non-English data better than English data while it was the opposite for both *LLM* and *LLM-FT*.

The number of unique tokens and their distribution throughout the file determine its entropy. Given this and considering the different compression ratios achieved by models on the English datasets, it is clear that a model's performance on one dataset of a language cannot be generalized to all documents in that language. Therefore, the compression ratios obtained from different datasets cannot be compared directly. For example comparing the LLM's compression ratios for *en* and *ar* does not, by itself, indicate whether it compresses Arabic texts worse than English texts in general. A more meaningful means of comparison is to evaluate the performance of the LLM relative to the best baseline on each dataset.

Table 4 indicates that the LLM's performance on English data was substantially closer to the strongest baseline than its performance on non-English data. For example, the weakest English result (*LLM-FT* on *en*) was approximately one-third worse than the best baseline, whilst the worst non-English result (*LLM* on *hindi*) was 256% poorer than the strongest baseline. Figure 1 demonstrates that the largest compression ratio differences belong to languages with completely different alphabet than English, except *ch*. This underlines LLM's weakness in modeling languages with thoroughly different characters than English.

Figure 2 shows that the LLM required dramatically more time to perform compression than baselines while the resulting compression ratios were

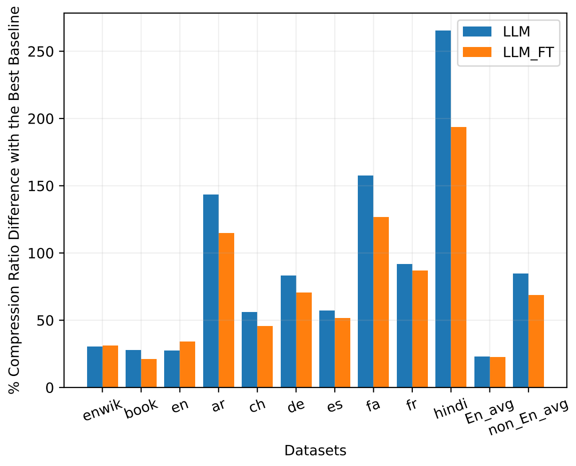

Figure 1: The percentage of the performance difference between the best baselines and the LLM. Fine-tuning significantly improved the compression performance of *LLM* on non-English data while its impact on English data was insignificant.

not comparable to the best baselines. On average, *LLM* required roughly 12 hours to compress 100 MB of data, while fine-tuning increased this by approximately 5 hours. Baselines, however, performed compressions within a few minutes. Similar to the LLM, *zlib*'s compression time for non-English data was longer than English, whereas *bz2, lzma,* and *pyppmd* were language-agnostic on average. However, Tawa models compressed non-English datasets quicker than English datasets. Unlike compression ratios, the LLM's compression duration of languages with common characters with English was the worst (except *ch*). Compression durations are included in Appendix.

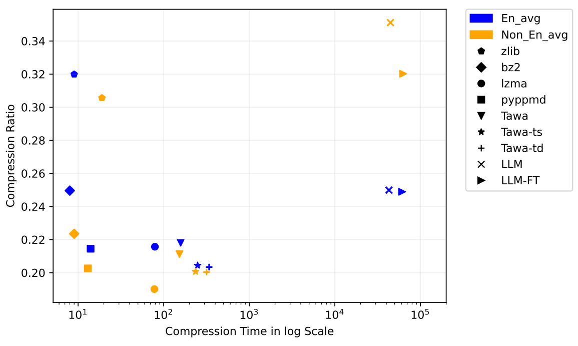

Figure 2: The time taken by each model to compress each dataset in log scale and the resulting compression ratios.

Both *LLM* and *LLM-FT* exhausted the available GPU cores and GPU memory regardless of the dataset being compressed. The GPU usage report is included in Appendix.

Model	enwik	book	en	ar	ch	de	es	fa	fr	hindi	En-Avg	Non-En-Avg
zlib	<u>0.3649</u>	<u>0.3591</u>	<u>0.2357</u>	0.2153	<u>0.4488</u>	<u>0.3771</u>	<u>0.3720</u>	0.2894	0.2726	0.1642	<u>0.3199</u>	0.3056
bz2	<u>0.2896</u>	<u>0.2759</u>	0.1832	0.1442	0.3527	0.2878	0.2948	0.1962	0.1951	0.0939	<u>0.2496</u>	0.2235
lzma	0.2478	0.2527	**0.1465**	**0.1323**	0.3017	0.2505	0.2562	**0.1330**	**0.1683**	**0.0883**	0.2157	**0.1900**
pyppmd	0.2478	0.2303	0.1654	0.1461	0.3163	0.2365	0.2476	0.1848	0.1697	0.1169	0.2145	0.2026
Tawa	0.2389	0.2334	0.1820	0.1791	0.2910	0.2281	0.2412	0.2026	0.1928	0.1435	0.2181	0.2112
Tawa-ts	0.2163	0.2238	0.1732	0.1780	0.2588	0.2138	0.2265	0.2017	0.1827	0.1433	0.2044	0.2007
Tawa-td	**0.2151**	**0.2220**	0.1731	0.1781	**0.2584**	**0.2133**	**0.2259**	0.2017	0.1823	0.1433	**0.2034**	0.2004
LLM	0.2799	0.2833	0.1864	0.3218	0.4033	0.3903	0.3552	0.3423	0.3226	0.3224	0.2499	0.3511
LLM-FT	0.2815	0.2687	0.1965	0.2840	0.3765	0.3635	0.3425	0.3015	0.3144	0.2592	0.2489	0.3202

Table 3: Compression ratios achieved by models on English and non-English datasets. On average, there is a noticeable gap between the compression performance of LLMs on English and non-English data. The best results are shown in **bold** and the baselines outperformed by the LLM are <u>underlined</u>.

Model	enwik	book	en	ar	ch	de	es	fa	fr	hindi	En-Avg	Non-En-Avg
LLM	30.13	27.61	27.24	143.21	56.08	82.98	57.25	157.37	91.68	265.12	22.84	84.76
LLM-FT	30.88	21.03	34.13	114.64	45.71	70.41	51.62	126.69	86.81	193.54	22.36	68.50

Table 4: Percentage of the compression ratio differences between *LLM* and *LLM-FT*, and the best baseline (shown in bold in table 3) on each dataset.

The LLM used slightly more CPU power than baselines. Although baselines' CPU usage varied negligibly for different datasets, that of *LLM* and *LLM-FT* remained almost the same. On average, there is an insignificant difference between the CPU utilization of baselines on English and non-English data, showing that CPU usage is almost data-agnostic. The percentages of CPU utilization are included in Appendix.

Figure 3 demonstrates that despite using significantly more memory than baselines, *LLM* and *LLM-FT* were unable to achieve competitive compression ratios. The LLM required less memory to compress non-English data than English data on average while they compressed non-English datasets worse. Similarly, Tawa models used more memory space on English data, while the rest of the baselines occupied more memory on non-English data. The impressive low memory usage of Tawa models, especially the trained static model, is noteworthy. Similar to the compression time, the memory usage of the LLM was worse on languages with shared characters with English, with the exception of *hindi*. Memory space usages are included in Appendix.

Figure 4 shows that *LLM* and *LLM-FT* were almost twice more precise on English data than on non-English data on average. The decrease in the performance on *enwik* and *en* after fine-tuning resulted in almost identical accuracies for *LLM-FT* and *LLM* on English datasets on average. However, the contribution of fine-tuning to the performance of *LLM* on non-English datasets was considerable. The witnessed trend in compression ratios, performing worse on languages with totally different characters, is observed here, too. Surprisingly, although the LLM was more accurate on some datasets than others, it achieved worse compression ratios. E.g. the percentage of top-15 predicted tokens by *LLM* for *ch* was almost twice as high as *fa* (30.52% Vs. 17.04%) while the compression ratio of *ch* was nearly 18% worse than *fa* (0.4033 against 0.3423). The same behavior is witnessed for the percentage of 0 ranked predicted tokens. Tables including percentages of 0 and top-15 ranked tokens are included in Appendix.

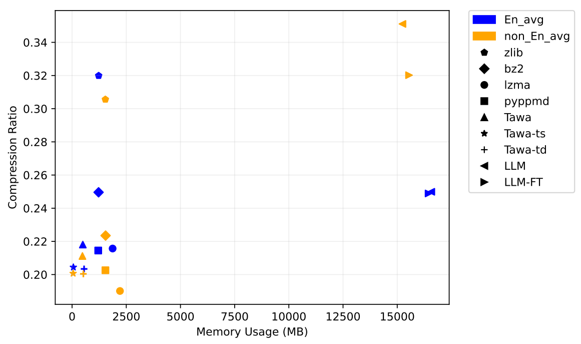

Figure 3: The memory used in compressing English and non-English datasets on average in MB against the resulting compression ratios.

4.2 Compressing *enwik, enwik-swap* and *enwik-sub*

Swapping words changes the arrangement of the data, affecting the performance of traditional algorithms relying on the statistics of the data. Table 5 shows that compression ratios achieved by

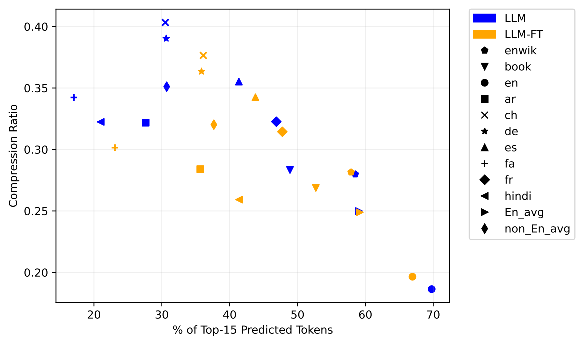

Figure 4: The percentage of top-15 predicted tokens by *LLM* and *LLM-FT* against the resulting compression ratios on each dataset.

traditional algorithms increased by between 7.48% (*zlib*) and 14.52% (*lzma*) on *enwik-swap*. Similarly, the compression performance of *LLM* and *LLM-FT* declined on the swapped data, with larger drops of 25.90% and 18.97%, respectively. As expected, the baselines' performance on *enwik-sub* almost remained constant since substituting characters does not change the number of unique characters and their frequencies. Nevertheless, figure 5 shows that compression ratios of the LLM on *enwik-sub* was approximately twice as high as the amounts it achieved on the original data, highlighting the reliance of LLMs on semantic and contextual relationships among the tokens.

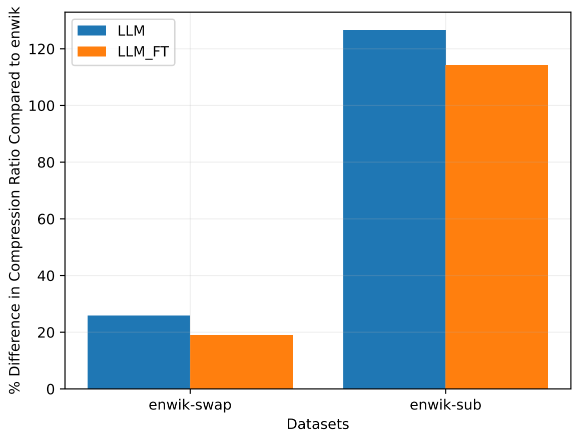

Figure 5: The percentage of the differences between the compression ratios achieved by *LLM* and *LLM-FT* on *enwik-swap* and *enwik-sub*, relative to *enwik*.

Figure 6 shows while the compression time of *enwik, enwik-swap* and *enwik-sub* by the LLM was dramatically more than baselines, it was unable to achieve comparable compression ratios. The LLM required more than 12 hours to compress *enwik-swap*, similar to the rest of the datasets. However,

Model	enwik	enwik-swap	enwik-sub
zlib	<u>0.3649</u>	<u>0.3922</u>	0.3649
bz2	<u>0.2896</u>	0.3172	0.2894
lzma	0.2478	0.2838	0.2477
pyppmd	0.2478	0.2752	0.2478
Tawa	0.2389	0.2622	0.2389
Tawa-ts	0.2163	0.2379	0.2163
Tawa-td	**0.2151**	**0.2369**	**0.2151**
LLM	0.2799	0.3524	0.6342
LLM-FT	0.2815	0.3349	0.6030

Table 5: The compression ratios of *enwik, enwik-swap*, and *enwik-sub* achieved by each model. The best results are shown in **bold** and the baselines surpassed by the LLM are <u>underlined</u>.

although the compression time of the substituted data was almost identical to the original dataset for baselines, the LLM required twice as much time to compress *enwik-sub* as *enwik*. Furthermore, although baselines' compression time for *enwik-swap* was longer than *enwik*, *LLM* and *LLM-FT* compressed it faster. Compression time reports are included in Appendix.

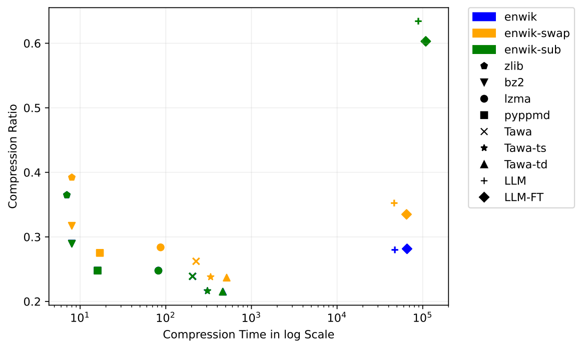

Figure 6: The time taken for each model in log scale to compress *enwik, enwik-swap* and *enwik-sub*. Since the results for *enwik* were almost identical to those for *enwik-sub* for baselines, their symbols overlap.

Similar to the previous experiments, the LLM exhausted both GPU power and GPU memory to compress the original, swapped and substituted datasets. The GPU usage details are in Appendix.

Analogous to the previous results, the CPU usage of the baselines varied slightly, whereas that of LLMs remained almost unchanged. Interestingly, Tawa models used more CPU to compress *enwik-sub* than *enwik* while the compression ratios were the same. The percentages of CPU usage of each compression are included in Appendix.

Figure 7 depicts that despite using significantly more memory, the LLM was unable to achieve compression ratios close to the best baselines. *Tawa-ts* showed significant efficiency among all baselines. Although the memory usage of Tawa models were slightly higher when compressing *enwik-swap* than compressing *enwik*, the rest of the models used less memory space to compress the swapped data. Tawa models used almost the same memory space to compress both *enwik* and *enwik-sub* while the rest of the models required considerably more memory to compress the latter. The memory usage table is included in Appendix.

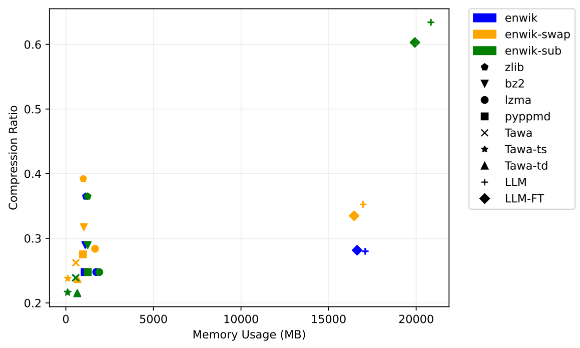

Figure 7: Memory consumption of models during compressing *enwik, enwik-swap*, and *enwik-sub*. Fine-tuning helped with the memory consumption of the LLM. Baselines' symbols for *enwik* and *enwik-sub* overlap as their memory usage was almost identical.

Figure 8 demonstrates that the percentage of the top-15 predicted tokens decreased in both swapped and substituted datasets comparing to the original dataset. Surprisingly, despite the close modeling accuracies achieved by the LLM on *enwik-sub* and *enwik-swap*, there is a wide gap between the resulting compression ratios. Also, the LLM was more accurate in predicting rank 0 tokens on *enwik-sub* than *enwik-swap*. The percentages for the rank 0 and top-15 predicted tokens are included in Appendix.

5 Discussion

Experiments show that although using the 4-bit-quantized LLaMA-3.2-1B through FineZip's approach to compress text data surpassed *zlib* on English datasets, it was outperformed by *lzma*, *pyppmd* and Tawa models. Considering non-English data, LLM models performed significantly worse than baselines. Comparing the compression ratios between the LLM and the best baselines shows *LLM* and *LLM-FT* performed 22.84% and 22.36% worse, respectively, on English data on average, whilst the figures for non-English datasets show 84.76% and 68.50% worse performance, respectively. This highlights that due to being trained mostly on English data, LLMs do not compress non-English data as well as English data, aligning with the findings of AlphaZip.

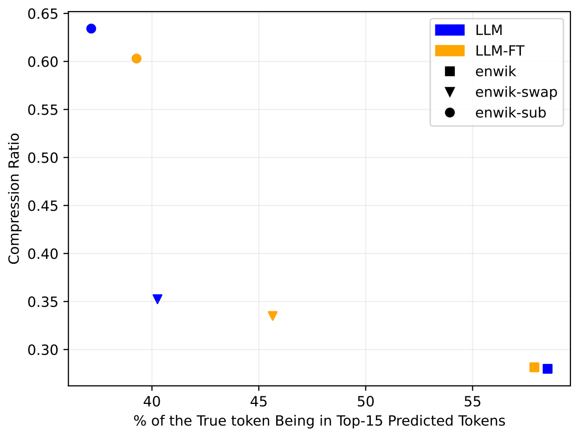

Figure 8: Percentage of the top-15 predicted tokens of *enwik, enwik-swap*, and *enwik-sub* by *LLM* and *LLM-FT*. Except for *enwik*, fine-tuning appeared effective.

Experiments on the performance of the LLM on the swapped and substituted datasets show that *LLM* and *LLM-FT* performed 25.90% and 18.97% worse on swapped data than they did on the normal dataset, respectively, while compression ratios of the substituted data were noticeably worse. The performance of *LLM* on *enwik-sub* was 126.58% worse than on its performance on *enwik*. The compression ratio of *LLM-FT* was slightly better, with being 114.21% worse than that of the original data. This casts serious doubts on LLMs' performance on unstructured text data, such as log files or encrypted documents.

Resource usage is the major demerit of LLMs. While baselines achieved low compression ratios in a few minutes, the LLM normally required approximately 12 (up to 25) hours to achieve inferior results. Also, Fine-tuning added roughly 4.5 hours overhead. The LLM exhausted the GPU by using the full capacity of its cores and memory, whilst baselines only used the CPU to compress. The LLM's CPU usage was identical for all compression and fine-tuning tasks, which is potentially due to the execution of the underlying Python code. Furthermore, while the LLM occupied at least five times more memory space than baselines, Tawa models were significantly memory efficient.

Given that the rankings predicted by the LLM

Model	enwik	enwik-swap	enwik-sub	book	en	ar	ch	de	es	fa	fr	hindi
Tawa-ts	184.22	192.73	184.22	120.18	122.73	88.05	241.09	147.70	148.47	86.46	128.38	84.26
Tawa-td	261.89	277.49	261.89	148.91	152.17	91.39	362.99	197.41	198.89	88.57	162.44	84.83
LLM	2,357.16	2,357.16	2,357.16	2,357.16	2,357.16	2,357.16	2,357.16	2,357.16	2,357.16	2,357.16	2,357.16	2,357.16

Table 6: The size of the base model (84 MB for Tawa and 2,357 MB for the LLM) plus the size of the trained/fine-tuned models required to decompress each dataset in MB.

are compressed in the FineZip's approach, there is a general inverse correlation between the LLM's modeling accuracy and the resulting compression ratios; i.e. the more accurate the LLM, the better compression ratio it achieves. However, our findings show the opposite between some datasets. E.g. the accuracy of the LLM on *es* was higher than that of *hindi*, whereas its compression ratio was worse.

Since each compression finally needs a decompression to obtain the original data, the same model used for the compression must be used to perform the decompression. This highlights the importance of the model size in real-world compression problems. The baselines are Python libraries occupying an insignificant amount of hard disk, and Tawa toolkit source code is approximately 84 MB. LLaMA-3.2-1B, however, requires 2,357 MB of hard disk. Table 6 shows that training a static Tawa model added smaller storage overhead than a dynamic model. Interestingly, the utilization of PEFT in the FineZip's "online memorization" step added a negligible amount of storage overhead (less than 20 KB). However, given the disk usage of the base models, which are required besides the trained/fine-tuned versions for decompression, LLaMA-3.2-1B is highly disadvantageous.

This work does not invalidate the findings of previous work in achieving superior compression ratios using larger LLMs. However, the question is whether it is reasonable or practical to use vast computational resources and excessive time to achieve SOTA compression ratios on just 10 Mega bytes of plain English data. LLMs perform extraordinarily well on a wide variety of tasks. But, our results show they are significantly worse than traditional baseline compressors in every aspect.

6 Future Work

Tawa toolkit and LZMA showed notably superior performance in both compression and resource usage. Focusing on their development will result in even better performance.

Developing language-specific neural network compressors is a more promising field of research than working on LLM compressors.

LLaMA-3.2-1B showed an unexpected behavior in yielding worse compression ratios when it had performed better in modeling the data. An investigation of FineZip's approach is needed to discover the root of this phenomenon.

7 Conclusion

This work shows LLMs' text compression performance significantly declines on both non-English and unstructured English data compared with plain English data. Through experiments using LLaMA-3.2-1B as the LLM within FineZip's framework to compress datasets from eight most widely spoken languages, we found that its performance on non-English datasets was more than three times worse than its performance on English datasets relative to the best baseline. Moreover, its compression performance on unstructured data was worse by a factor of two compared to plain English data. In contrast, traditional baselines performed nearly consistently across datasets. Surprisingly, although the LLM modeled some datasets more accurately than others, the resulting compression ratios were poorer. Furthermore, the LLM utilized all available GPU power and memory, occupied at least five times more memory space than baselines, and required 12 hours on average to complete compression tasks, whereas baselines completed the same tasks within minutes by utilizing significantly fewer resources. These findings raise serious questions about the practicality of current LLMs for data compression, where traditional approaches appear reliable.

Limitations

Despite the first aspirations to study more powerful LLMs, limited resources forced us to resort to a small model. In fact, LLaMA-3.2-1B does not fit into the 16GB memory of an NVIDIA P100 GPU without 4-bit quantization. Larger LLMs without quantization can achieve superior compression performance as shown in previous work.

Relying on only one dataset per each non-English language may make language-wise results inconclusive. This work, nevertheless, discussed the performance of models on non-English datasets as a whole, a large set consisting of datasets from seven most widely spoken languages.

Conclusions on the compression performance of LLMs on non-English and unstructured data have been reached based on experiments using only one LLM. However, our findings are generalizable to other models considering that plain English text comprises the dominant part of the training data of LLMs.

Acknowledgment

We acknowledge the support of the Supercomputing Wales project, which is part-funded by the European Regional Development Fund (ERDF) via Welsh Government.

References

Hadi Abdi Khojasteh, Ebrahim Ansari, and Mahdi Bohlouli. 2020. LSCP: Enhanced large scale colloquial Persian language understanding. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 6323–6327, Marseille, France. European Language Resources Association. License: CC BY-NC-ND 4.0.

Jyrki Alakuijala, Andrea Farruggia, Paolo Ferragina, Eugene Kliuchnikov, Robert Obryk, Zoltan Szabadka, and Lode Vandevenne. 2018. Brotli: A general-purpose data compressor. *Association for Computing Machinery*, 37(1).

Fabrice Bellard. 2019. NNCP: Lossless data compression with neural networks. `https://bellard.org/nncp/`. Accessed: 2025-08-12.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Barry Haddow, Matthias Huck, Chris Hokamp, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Carolina Scarton, Lucia Specia, and Marco Turchi. 2015. Findings of the 2015 workshop on statistical machine translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 1–46, Lisbon, Portugal. License: ODbL v1.0.

Umar Butler. 2025. Open australian legal corpus. Isaacus, license: CC BY 4.0.

David Cox. 2016. Syntactically informed text compression with recurrent neural networks. *Preprint*, arXiv:1608.02893.

Gregoire Deletang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. 2024. Language modeling is compression. In *The Twelfth International Conference on Learning Representations*.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science*, 14(2):179–211.

Jean-loup Gailly. Gzip. `https://www.gzip.org/`. Accessed: 2025-08-24.

Henry Gilbert, Michael Sandborn, Douglas C. Schmidt, Jesse Spencer-Smith, and Jules White. 2023. Semantic compression with large language models. In *2023 Tenth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 1–8.

David Gili Fernández De Romarategui. 2024. Compressing network data with deep learning. Master's thesis, Facultat d'Informàtica de Barcelona (FIB), Universitat Politècnica de Catalunya (UPC) - BarcelonaTec.

Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. 2018. DeepZip: Lossless data compression using recurrent neural networks. *2019 Data Compression Conference (DCC)*, pages 575–575.

Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. 2021. DZip: improved general-purpose loss less compression based on novel neural network modeling. In *2021 Data Compression Conference (DCC)*, pages 153–162.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The Llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Asier Gutiérrez-Fandiño, David Pérez-Fernández, Jordi Armengol-Estapé, David Griol, and Zoraida Callejas. 2022. escorpius: A massive spanish crawling corpus. In *IberSPEECH 2022*, pages 126–130. License: CC BY-NC-ND 4.0.

David Heurtel-Depeiges, Anian Ruoss, Joel Veness, and Tim Genewein. 2025. Compression via pre-trained transformers: A study on byte-level multimodal data. In *Forty-second International Conference on Machine Learning*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Cynthia Huang, Yuqing Xie, Zhiying Jiang, Jimmy Lin, and Ming Li. 2023. Approximating human-like few-shot learning with GPT-based compression. *ArXiv*, abs/2308.06942.

Yuzhen Huang, Jinghan Zhang, Zifei Shan, and Junxian He. 2024. Compression represents intelligence linearly. In *First Conference on Language Modeling*.

David A. Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

Byron Knoll. 2014. CMIX. `https://www.byronknoll.com/cmix.html`. Accessed: 2025-08-12.

Byron Knoll. 2015. lstm-compress. `https://github.com/byronknoll/lstm-compress`. Accessed: 2025-08-12.

Byron Knoll. 2017. tensorflow-compress. `https://github.com/byronknoll/tensorflow-compress`. Accessed: 2025-08-12.

Ziguang Li, Chao Huang, Xuliang Wang, Haibo Hu, Cole Wyeth, Dongbo Bu, Quan Yu, Wen Gao, Xingwu Liu, and Ming Li. 2025. Lossless data compression by large models. *Nature Machine Intelligence*, 7(5):794–799.

Qian Liu, Yiling Xu, and Zhu Li. 2019. DecMac: A deep context model for high efficiency arithmetic coding. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 438–443.

Huidong Ma, Hui Sun, Liping Yi, Yanfeng Ding, Xiaoguang Liu, and Gang Wang. 2025. MSDZip: Universal lossless compression for multi-source data via stepwise-parallel and learning-based prediction. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, page 3543–3551, New York, NY, USA. Association for Computing Machinery.

Matt Mahoney. 2006. Large text compression benchmark. `https://mattmahoney.net/dc/textdata.html`. Accessed: 2025-08-12.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. PEFT: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`.

Yu Mao, Yufei Cui, Tei-Wei Kuo, and Chun Jason Xue. 2022a. Accelerating general-purpose lossless compression via simple and scalable parameterization. In *Proceedings of the 30th ACM International Conference on Multimedia*, MM '22, page 3205–3213, New York, NY, USA. Association for Computing Machinery.

Yu Mao, Yufei Cui, Tei-Wei Kuo, and Chun Jason Xue. 2022b. TRACE: A fast transformer-based general-purpose lossless compressor. In *Proceedings of the ACM Web Conference 2022*, WWW '22, page 1829–1838, New York, NY, USA. Association for Computing Machinery.

Yu Mao, Jingzong Li, Yufei Cui, and Jason Chun Xue. 2023. Faster and stronger lossless compression with optimized autoregressive framework. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6.

Meta-AI. 2024. meta-llama/Llama-3.2-1B · Hugging Face. `https://huggingface.co/meta-llama/Llama-3.2-1B`. LiceAccessed: 2025-09-07.

Fazal Mittu, Yihuan Bu, Akshat Gupta, Ashok Devireddy, Alp Eren Ozdarendeli, Anant Singh, and Gopala Anumanchipalli. 2024. FineZip : Pushing the limits of large language models for practical lossless text compression. *Preprint*, arXiv:2409.17141.

Hiroshi Miura. 2021. PyPPMd. `https://pyppmd.readthedocs.io/en/latest/`. Accessed: 2025-09-03.

Muennighoff. 2021. bookcorpus. `https://www.kaggle.com/datasets/muennighoff/bookcorpus`. License: CC0 1.0, Accessed: 2025-09-02.

MultiSynt. 2025. Mt-nemotron-cc: Large-scale machine-translated high quality web text. A translated variant of Nemotron-CC High Quality for multilingual LLM pretraining, License: ODC-By v1.0.

Swathi Shree Narashiman and Nitin Chandrachoodan. 2024. AlphaZip: Neural network-enhanced lossless text compression. *Preprint*, arXiv:2409.15046.

OpenAI. 2022. Introducing ChatGPT. `https://openai.com/index/chatgpt/`. Accessed: 2025-08-13.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Gv, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, and 13 others. 2023. RWKV: Reinventing RNNs for the transformer era. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14048–14077, Singapore.

Python Standard Library. a. bz2. `https://docs.python.org/3/library/bz2.html`. Accessed: 2025-09-03.

Python Standard Library. b. lzma. `https://docs.python.org/3/library/lzma.html`. Accessed: 2025-09-03.

Python Standard Library. c. zlib. `https://docs.python.org/3/library/zlib.html`. Accessed: 2025-09-03.

Claude E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423.

Atul Kumar Singh. 2024. hindi-TinyStories. Hugging Face Datasets, Accessed: 2025-09-10.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. 2025. Nemotron-CC: Transforming Common Crawl into a refined long-horizon pretraining dataset. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2459–2475, Vienna, Austria. License: ODC-By v1.0.

Supercomputer-Wales. 2025. About Hawk – Supercomputing Wales Portal. `https://portal.supercomputing.wales/index.php/about-hawk/`. Accessed: 2025-09-10.

William Teahan. 2018. A compression-based toolkit for modelling and processing natural language text. *Information*, 9(294):1–29.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. LLaMA: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023b. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Chandra Shekhara Kaushik Valmeekam, Krishna Narayanan, Dileep Kalathil, Jean-Francois Chamberland, and Srinivas Shakkottai. 2023. LLMZip: Lossless text compression using large language models. *Preprint*, arXiv:2306.04050.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Jiawei Wang and Qingxin Zhang. 2024. ALCZip: Fully exploitation of large models in lossless text compression. In *2024 4th International Conference on Electronic Information Engineering and Computer Communication (EIECC)*, pages 1007–1012.

Liangdong Wang, Bo-Wen Zhang, Chengwei Wu, Hanyu Zhao, Xiaofeng Shi, Shuhao Gu, Jijie Li, Quanyue Ma, TengFei Pan, and Guang Liu. 2024. CCI3.0-HQ: a large-scale chinese dataset of high quality designed for pre-training large language models. *Preprint*, arXiv:2410.18505. License: apache-2.0.

Ian H. Witten, Radford M. Neal, and John G. Cleary. 1987. Arithmetic coding for data compression. *Commun. ACM*, 30(6):520–540.

Sane Yagi and Ashraf Elnagar. 2024. Arabic punctuation dataset. Mendeley Data, license: CC BY 4.0.

Junxuan Zhang, Zhengxue Cheng, Yan Zhao, Shihao Wang, Dajiang Zhou, Guo Lu, and Li Song. 2025. L3TC: Leveraging RWKV for learned lossless low-complexity text compression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(12):13251–13259.

Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 19–27.

Emir Öztürk and Altan Mesut. 2024. Learning-based short text compression using BERT models. *PeerJ Computer Science*, 10.

A Models' Parameters

Table 7 provides the parameter setting of each model.

Model	**Parameters**
zlib	`level=9`
bz2	`compresslevel=9`
lzma	`preset=9`
pyppmd	*Default Parameters*
Tawa	Alphabet size set to 256, escape method set to D, PPM order set to 5 (`-a 256 -e D -O 5`)
Tawa-ts	Alphabet size set to 256, escape method set to D, PPM order set to 5 (`-a 256 -e D -O 5`)
Tawa-td	Alphabet size set to 256, escape method set to D, PPM order set to 5 (`-a 256 -e D -O 5`)
LLM	`contex_size=512`, `batch_size=64`
LLM-FT	`block_size=128 epochs=256, r=8, learning_rate=1e-4, batch_size=64` for fine-tuning and `contex_size=512` and `batch_size=64` for compression

Table 7: Summary of models

B Datasets

To obtain *enwik*, the first 100 MB of The Enwik 9 dataset (Enwik 8) was extracted with `head -c 100M` Unix command. For *enwik-swap*, the list of words of *enwik* was first obtained. Then, each word with even index in the list (0, 2, 4...) was replaced by the word located two positions ahead, while each word with odd index was swapped with the word four positions ahead. With this swapping method, for example, the sentence "Albert Einstein was born in Ulm." will become "was Ulm. in Einstein Albert born". For *enwik-sub*, the ASCII code of each English lowercase and uppercase character of *enwik* was substituted with the code 13 positions ahead. For instance, the characters of the word "Compression!" was first transformed to their

ASCII code (67-111-109-112-114-101-115-115-105-111-110-33). Next, each English character was replaced with the code 13 positions ahead (80-98-122-99-101-114-102-102-118-98-97-33). Finally, the ASCII code was translated to characters "Pbzcerffvba!". Remember the range of ASCII code of uppercase English characters is between 65 to 90 and lowercase characters is between 97 to 122. For *book*, the first 10 MB of the BookCorpus, which was downloaded from Kaggle (Muennighoff, 2021), was excluded as it contains bit representations, which makes it unreadable by Python's `read` function. For *en*, values of the `text` feature of the Open Australian Legal Corpus were extracted through the HugginFace's `datasets` Python module. For *ar*, the content of the first three text files of the Arabic Punctuation Dataset were concatenated in a 122 MB file. For *ch*, values of the `content` feature of the second `JSON` file of the dataset were read and stored in a text file since the first `JSON` file was unreadable by Python's `read` function due to containing bit representations. For *de*, the first `parquet` file of the German part of the Nemotron dataset was downloaded through the HuggingFace's `datasets` Python module and the values of the `text` feature were stored in a text file. For *es*, the first `JSON` file of the dataset was downloaded and the `text` feature of each sample was extracted through Regular Expressions and stored in a text file. For *fa*, the Farsi-English file of the LSCP dataset was downloaded and the English characters were removed via Regular Expression. For *fr*, the values of the `fr` feature of the French-English translation dataset were stored in a text file. For *hindi*, the values of the `text` feature of the first `parquet` file of the Hindi-TinyStories were stored in a text file. All the mentioned approaches stored data in files with `utf-8` encoding. Ultimately, `head -c 100M` Unix command was applied on all the resulted files, except for *enwik, enwik-swap* and *enwik-sub*, to extract the first 100 MB of them.

C Statistics Tables

It is worth noting that the figures for *enwik-swap* and *enwik-sub* are not counted towards the **En-avg** values.

C.1 Training/Fine-tuning Statistics

Table 8 compares the LLM's fine-tuning and Tawa models' training processes duration. On average, fine-tuning time was dataset-agnostic. Table 9 shows the average percentage of CPU usage of each fine-tuning/training process. Overall, the CPU usage of these processes was almost dataset-agnostic. Table 10 compares the memory usage of fine-tuning/training processes. While Tawa models used more memory on English data on average, it was vice versa for the LLM. Interestingly, Tawa models used substantially more memory during their training than compressing while the LLM used significantly less memory for fine-tuning than for compressing. Table 11 shows the average percentage of GPU power and memory used by the LLM's fine-tuning process. It exhausted the GPU regardless of the dataset. The identical GPU memory usage for all datasets is noteworthy.

C.2 Compression Statistics

Table 12 compares the time required to compress each dataset by all models. Table 13 shows the LLM occupied almost all the available GPU memory regardless of the dataset being compressed. Table 14 shows that although the LLM consumed slightly less GPU power on non-English data than on English data on average, it saturated the GPU cores in all cases. Also, Fine-tuning slightly reduced the GPU usage in most of the datasets. Table 15 compares the CPU usage of each compression process. Table 16 presents the memory usage of models while compressing datasets. Table 17 shows fine-tuning improved the accuracy of the LLM on all datasets, except *enwik, en* and *fr*. While both *LLM* and *LLM-FT* modeled *enwik-sub* more accurately than *enwik-swap*, the compression ratios of the substituted data were considerably worse than those of the swapped data. Table 18 compares the percentage of the top-15 tokens predicted by the LLM on each dataset.

Model	enwik	enwik-swap	enwik-sub	book	en	ar	ch	de	es	fa	fr	hindi	En-avg	non-En-avg
Tawa-ts	197	220	198	125	127	72	303	187	185	71	134	63	150	145
Tawa-td	207	218	102	126	127	72	304	187	184	71	134	64	153	145
LLM-FT	17,168	17,124	17,499	17,209	17,121	17,106	17,201	17,219	17,163	17,169	17,180	17,494	17,166	17,219

Table 8: LLM fine-tuning and Tawa training time in seconds.

Model	enwik	enwik-swap	enwik-sub	book	en	ar	ch	de	es	fa	fr	hindi	En-avg	non-En-avg
Tawa-ts	2.21	2.22	2.23	2.21	2.19	2.17	2.21	2.19	2.20	2.16	2.23	2.19	2.20	2.19
Tawa-td	2.22	2.22	2.18	2.22	2.23	2.25	2.20	2.19	2.18	2.23	2.22	2.23	2.22	2.21
LLM-FT	2.44	2.44	2.45	2.44	2.44	2.44	2.44	2.44	2.44	2.44	2.44	2.45	2.44	2.44

Table 9: The average percentage CPU usage of the LLM fine-tuning and Tawa training processes.

Model	enwik	enwik-swap	enwik-sub	book	en	ar	ch	de	es	fa	fr	hindi	En-avg	non-En-avg
Tawa-ts	664.03	680.79	664.03	468.71	503.70	406.12	812.65	564.83	570.39	403.46	518.32	399.88	545.48	525.09
Tawa-td	1,376.72	1,378.29	1,377.54	1,176.18	1,194.55	913.03	1,475.19	1,287.07	1,298.17	404.75	1,151.87	401.68	1,249.15	990.25
LLM-FT	3,652.30	3,636.90	4,321.54	3,640.36	3,478.67	3,558.90	3,859.10	3,721.42	3,725.34	3,527.42	3,705.27	3,532.16	3,590.44	3,661.37

Table 10: The memory usage of the LLM fine-tuning and Tawa training processes in MB

Dataset	GPU Avg (%)	GPU Mem (MB)
enwik	99.85	14,956
enwik-swap	99.86	14,956
enwik-sub	99.87	14,956
book	99.85	14,956
en	99.86	14,956
ar	99.85	14,956
ch	99.85	14,956
de	99.85	14,956
es	99.85	14,956
fa	99.85	14,956
fr	99.85	14,956
hindi	99.87	14,956
En-Avg	99.85	14,956
Non-En-Avg	99.85	14,956

Table 11: The GPU power (%) and memory (MB) usage of each LLM fine-tuning process.

Model	enwik	enwik-swap	enwik-sub	book	en	ar	ch	de	es	fa	fr	hindi	En-Avg	Non-En-Avg
zlib	**7**	**8**	**7**	11	**8**	33	**8**	10	10	30	**8**	36	9	19
bz2	8	**8**	8	**9**	**8**	**10**	**8**	**9**	**9**	11	9	**8**	**8**	**9**
lzma	82	87	82	93	64	74	82	93	94	68	70	68	79	78
pyppmd	16	17	16	14	12	**10**	19	15	15	**10**	12	10	14	13
Tawa	206	226	205	132	136	78	311	193	194	81	142	73	158	153
Tawa-ts	306	334	307	213	227	139	418	294	300	137	230	131	249	236
Tawa-td	465	514	462	280	276	155	630	419	433	153	293	139	340	317
LLM	47,278	46,450	88,848	42,508	38,858	38,778	54,172	49,141	48,814	37,112	48,006	36,598	42,882	44,660
LLM-FT	65,594	64,741	108,522	60,726	56,924	56,842	72,663	67,567	67,144	55,144	66,366	55,005	61,081	62,962

Table 12: The time taken by models to compress each dataset in seconds (training/fine-tuning time + compression time for *Tawa-ts*, *Tawa-td* and *LLM-FT*). The best results are shown in bold.

Model	enwik	enwik-swap	enwik-sub	book	en	ar	ch	de	es	fa	fr	hindi	En-Avg	Non-En-Avg
LLM	15,956	16,066	16,158	16,160	14,516	16,142	15,982	16,068	16,170	16,166	16,060	13,674	15,544	15,752
LLM-FT	16,270	16,202	16,112	15,828	16,176	14,264	15,824	16,172	16,068	13,846	15,842	16,254	16,091	15,467

Table 13: GPU memory consumption by the LLM during each compression in MB. The GPU memory usage of the fine-tuning steps is not considered in this table.

Model	enwik	enwik-swap	enwik-sub	book	en	ar	ch	de	es	fa	fr	hindi	En-Avg	Non-En-Avg
LLM	98.60	98.62	98.69	98.58	98.59	98.23	98.65	98.64	98.61	98.59	98.24	98.60	98.59	98.51
LLM-FT	98.36	98.38	98.45	98.35	98.34	98.24	98.41	98.40	98.39	98.37	98.24	98.35	98.35	98.34

Table 14: The average percentage of GPU usage by *LLM* and *LLM-FT* when compressing each dataset. The GPU usage of fine-tuning processes is not considered in this table.

Model	enwik	enwik-swap	enwik-sub	book	en	ar	ch	de	es	fa	fr	hindi	En-Avg	Non-En-Avg
zlib	2.26	2.28	2.26	2.32	2.28	2.39	2.29	2.30	2.32	2.40	2.27	2.40	2.29	2.34
bz2	2.28	2.29	2.28	2.29	2.29	2.30	2.28	2.29	2.30	2.32	2.30	2.27	2.29	2.29
lzma	2.41	2.41	2.41	2.42	2.41	2.41	2.41	2.42	2.42	2.41	2.41	2.41	2.41	2.41
pyppmd	2.35	2.36	2.35	2.34	2.33	2.30	2.37	2.34	2.35	2.31	2.33	2.30	2.34	2.33
Tawa	2.21	2.21	**2.22**	2.23	**2.19**	2.18	2.22	2.22	**2.19**	2.23	2.20	2.25	**2.21**	2.21
Tawa-ts	**2.16**	**2.19**	2.23	2.23	2.23	**2.16**	**2.19**	**2.18**	2.22	**2.14**	**2.16**	**2.15**	**2.21**	**2.17**
Tawa-td	2.21	2.21	**2.22**	**2.22**	2.21	2.25	2.20	2.21	2.22	2.16	2.22	2.21	**2.21**	2.21
LLM	2.44	2.44	2.45	2.44	2.44	2.44	2.44	2.44	2.44	2.44	2.44	2.45	2.44	2.44
LLM-FT	2.44	2.44	2.45	2.44	2.44	2.44	2.44	2.44	2.44	2.44	2.44	2.45	2.44	2.44

Table 15: Average percentage of CPU usage by models when compressing each dataset. The best results are shown in bold.

Model	enwik	enwik-swap	enwik-sub	book	en	ar	ch	de	es	fa	fr	hindi	En-avg	non-En-avg
zlib	1,143.11	995.88	1,248.36	1,675.02	861.26	1,484.38	1,777.40	1,379.04	1,973.11	1,840.97	1,538.89	754.22	1,226.46	1,535.43
bz2	1,106.50	1,027.07	1,248.37	1,675.02	887.46	1,484.39	1,777.40	1,379.04	1,973.11	1,869.92	1,566.54	771.40	1,222.99	1,545.97
lzma	1,734.71	1,668.77	1,909.32	2,309.98	1,566.64	2,157.28	2,450.29	2,009.62	2,610.51	2,543.21	2,241.67	1,452.74	1,870.44	2,209.33
pyppmd	1,074.48	977.06	1,249.86	1,647.92	897.39	1,484.41	1,755.92	1,348.62	1,949.48	1,870.32	1,575.01	783.98	1,206.60	1,538.25
Tawa	562.30	571.35	563.33	459.47	464.35	407.16	654.70	504.36	504.11	402.70	473.31	401.73	495.37	478.30
Tawa-ts	**101.42**	**109.94**	**101.42**	**37.39**	**39.94**	**5.25**	**158.30**	**64.90**	**65.67**	**3.67**	**45.58**	**1.46**	**59.58**	**49.26**
Tawa-td	654.58	670.16	654.33	515.71	491.42	409.88	789.83	561.81	565.32	405.62	508.92	400.63	553.90	520.29
LLM	17,092.51	16,972.68	20,839.81	16,289.27	16,328.15	13,707.93	13,907.39	16,592.35	16,525.85	13,963.83	16,035.92	15,946.09	16,569.98	15,239.91
LLM-FT	16,629.01	16,456.50	19,928.51	16,695.30	16,018.47	14,381.92	14,722.00	15,999.87	17,072.24	14,689.84	16,600.34	15,356.34	16,447.59	15,546.08

Table 16: The memory space occupied by models when compressing each dataset in MB. The best results are shown in bold.

Model	enwik	enwik-swap	enwik-sub	book	en	ar	ch	de	es	fa	fr	hindi	En-Avg	Non-En-Avg
LLM	30.33	17.24	19.83	16.52	38.43	9.96	12.95	12.34	16.27	4.86	21.65	6.11	28.43	12.02
LLM-FT	29.50	19.76	21.94	18.71	35.43	13.38	16.13	14.11	17.87	6.84	21.09	13.85	27.88	14.75

Table 17: The percentage of correctly predicted tokens by the LLM on each dataset.

Model	enwik	enwik-swap	enwik-sub	book	en	ar	ch	de	es	fa	fr	hindi	En-Avg	Non-En-Avg
LLM	58.51	40.26	37.16	48.89	69.77	27.62	30.52	30.65	41.36	17.04	46.88	21.00	59.06	30.72
LLM-FT	57.89	45.65	39.28	52.70	66.94	35.67	36.12	35.85	43.80	23.09	47.77	41.39	59.18	37.67

Table 18: The percentage of top-15 tokens predicted by the LLMs on each dataset.