Language Modeling Is Compression

Grégoire Delétang[*1], Anian Ruoss[*1], Paul-Ambroise Duquenne[2], Elliot Catt[1], Tim Genewein[1], Christopher Mattern[1], Jordi Grau-Moya[1], Li Kevin Wenliang[1], Matthew Aitchison[1], Laurent Orseau[1], Marcus Hutter[1] and Joel Veness[1]
[*]Equal contributions, [1]Google DeepMind, [2]Meta AI & Inria

It has long been established that predictive models can be transformed into lossless compressors and vice versa. Incidentally, in recent years, the machine learning community has focused on training increasingly large and powerful self-supervised (language) models. Since these large language models exhibit impressive predictive capabilities, they are well-positioned to be strong compressors. In this work, we advocate for viewing the prediction problem through the lens of compression and evaluate the compression capabilities of large (foundation) models. We show that large language models are powerful general-purpose predictors and that the compression viewpoint provides novel insights into scaling laws, tokenization, and in-context learning. For example, Chinchilla 70B, while trained primarily on text, compresses ImageNet patches to 43.4% and LibriSpeech samples to 16.4% of their raw size, beating domain-specific compressors like PNG (58.5%) or FLAC (30.3%), respectively. Finally, we show that the prediction-compression equivalence allows us to use any compressor (like gzip) to build a conditional generative model.

1. Introduction

Information theory and machine learning are inextricably linked and have even been referred to as "two sides of the same coin" (MacKay, 2003). One particularly elegant connection is the essential equivalence between probabilistic models of data and lossless compression. The source coding theorem (Shannon, 1948) is the fundamental theorem describing this idea, i.e., the expected message length in bits of an optimal entropy encoder is equal to the negative $\log_2$-likelihood of the statistical model. In other words, maximizing the $\log_2$-likelihood (of the data) is equivalent to minimizing the number of bits required per message. Indeed, lossless compression with a probabilistic model can be achieved in a variety of different ways, including Huffman coding (Huffman, 1952), arithmetic coding (Pasco, 1977; Rissanen, 1976), and asymmetric numeral systems (Duda, 2009).

Arithmetic coding, in particular, is known to be optimal in terms of coding length, meaning that the overall compression performance depends on the capabilities of the probabilistic model (Fig. 1). Incidentally, in recent years, large pre-trained Transformers (Vaswani et al., 2017), so-called *foundation models* (Bommasani et al., 2021), have proven to be highly successful across a wide range of predictive tasks (Bubeck et al., 2023; Rae et al., 2021) and are thus promising candidates for use with arithmetic coding. Indeed, Transformer-based compression with arithmetic coding has produced state-of-the-art results both in the online (Bellard, 2021; Mao et al., 2022) and offline settings (Valmeekam et al., 2023). In the online setting, a pseudo-randomly initialized model is directly trained on the stream of data that is to be compressed, while the offline setting, which we consider in our work, trains the model on an external dataset before employing it to compress a (potentially different) data stream. Consequently, offline compression is performed *in-context*, with a fixed set of model parameters. Transformers have demonstrated impressive in-context learning abilities (Brown et al., 2020; Genewein et al., 2023; Laskin et al., 2023; Wei et al., 2022), which renders them ideally suited for offline compression. However, as we will discuss in this work, Transformers are actually trained to compress well, and therefore *must* have good in-context learning abilities.

Corresponding authors: {gdelt, anianr}@google.com

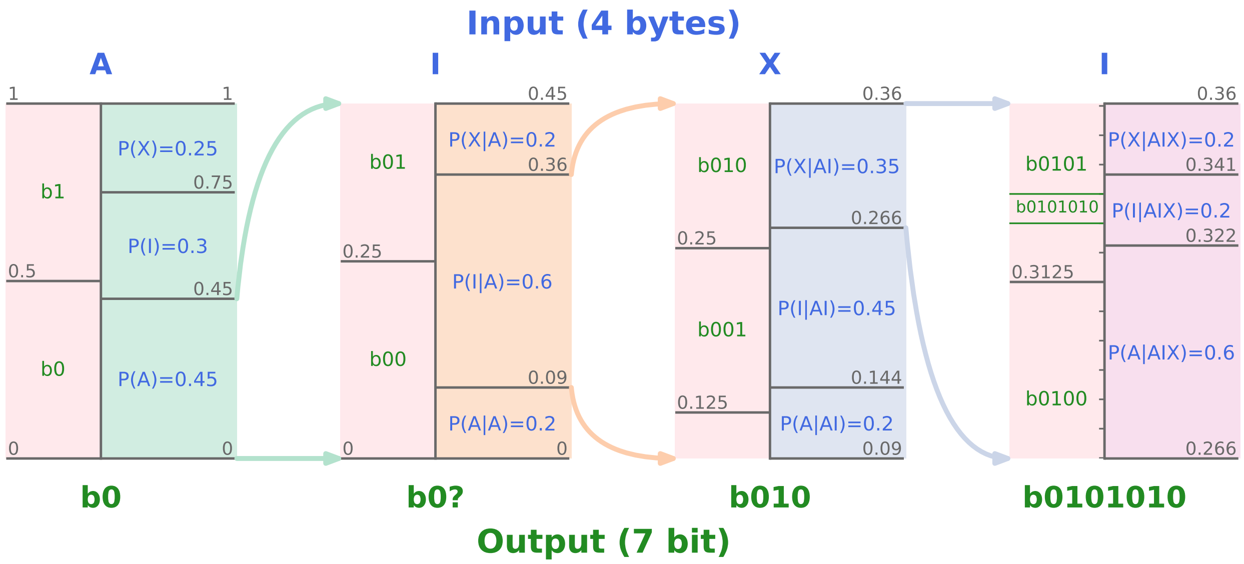


Figure 1 | Arithmetic encoding of the sequence 'AIXI' with a probabilistic (language) model P (both in blue) resulting in the binary code '0101001' (in green). Arithmetic coding compresses data by assigning unique intervals to symbols based on the probabilities assigned by P. It progressively refines these intervals to output compressed bits, which represent the original message. To decode, arithmetic coding initializes an interval based on the received compressed bits. It iteratively matches intervals with symbols using the probabilities given by P to reconstruct the original message.

The context length is a key limiting factor in offline compression, as it dictates the maximum number of bytes a model can compress at a time. Transformers can only compress a few kilobytes (each "token" being coded with 2 or 3 bytes), while requiring a lot of compute. Correspondingly, many challenging predictive tasks (e.g., algorithmic reasoning or long-term memory) require long contexts (Delétang et al., 2023), and thus extending these models' context lengths is a key challenge which is gaining increased attention (Bulatov et al., 2023; Guo et al., 2022; Zaheer et al., 2020). The in-context compression view provides insights into the failure modes of current foundation models.

This Work We advocate for using (lossless) compression to study foundation models. To that end, we conduct an extensive empirical investigation of the offline (in-context) compression capabilities of large language models, with the rationale that they have recently become readily available (Hoffmann et al., 2022; Touvron et al., 2023) and can thus be used for compression without the training overhead. We empirically demonstrate that these models, while (meta-)trained primarily on text, also achieve state-of-the-art compression rates across different data modalities, using their context to condition a general-purpose compressor to excel at a particular task. Moreover, we shed new light on scaling laws (Kaplan et al., 2020), showing that they also hold true for compression but that measuring the compression rates instead of the log loss adds a twist: Scaling beyond a certain point will deteriorate the compression performance since the model parameters need to be accounted for in the compressed output. Finally, we advocate for framing (self-supervised) prediction through the lens of compression as it encompasses generalization: a model that compresses well generalizes well (Hutter, 2006).

Contributions We make the following contributions:

- We empirically investigate the lossless compression capabilities of foundation models. To that end, we review how to compress with predictive models via arithmetic coding and call attention to the connection between current language modeling research and compression.
- We show that foundation models, trained primarily on text, are general-purpose compressors due to their in-context learning abilities. For example, Chinchilla 70B achieves compression rates of 43.4% on ImageNet patches and 16.4% on LibriSpeech samples, beating domain-specific compressors like PNG (58.5%) or FLAC (30.3%), respectively.
- We provide a novel view on scaling laws, showing that the dataset size provides a hard limit on model size in terms of compression performance and that scaling is not a silver bullet.
- We leverage the compression-prediction equivalence to employ compressors as generative models and visually illustrate the performance of the underlying compressor.
- We demonstrate that tokenization, which can be viewed as a pre-compression, does, in general, not improve compression performance, but allows models to increase the information content in their context and is thus generally employed to improve prediction performance.

2. Background

In this section, we review the necessary background on information theory and its relation to likelihood maximization. To that end, we consider streams of data $x_{1:n} := x_1 x_2 \dots x_n \in \mathcal{X}^n$ of length n from a finite set of symbols $\mathcal{X}$. We write $x_{\leq j} = x_{<j+1} := x_{1:j}$ for $j \leq n$ and denote the empty string as ϵ. Finally, we denote the concatenation of two strings s and r by sr.

Coding Distributions A coding distribution ρ is a sequence of probability mass functions $\rho_n : \mathcal{X}^n \mapsto (0, 1]$, which for all $n \in \mathbb{N}$ satisfy the constraint that $\rho_n(x_{1:n}) = \sum_{y \in \mathcal{X}} \rho_{n+1}(x_{1:n} y)$ for all $x_{1:n} \in \mathcal{X}^n$, with the base case $\rho_0(\epsilon) := 1$. From here on out, whenever the meaning is clear from the argument to ρ, we drop the subscript on ρ. Under this definition, the conditional probability of a symbol x_n given previous data $x_{<n}$ is defined as $\rho(x_n \mid x_{<n}) := \rho(x_{1:n})/\rho(x_{<n})$, with the familiar chain rules $\rho(x_{1:n}) = \prod_{i=1}^{n} \rho(x_i \mid x_{<i})$ and $\rho(x_{j:k} \mid x_{<j}) = \prod_{i=j}^{k} \rho(x_i \mid x_{<i})$ following.

Lossless Compression The goal of lossless compression is to encode a stream of symbols $x_{1:n}$ sampled from a coding distribution ρ into a bitstream of minimal (expected) length, while ensuring that the original data sequence is recoverable from the bitstream. To that end, we use a binary source code $c : \mathcal{X}^* \mapsto \{0, 1\}^*$, which assigns to each possible data sequence $x_{1:n}$ a binary code word $c(x_{1:n})$ of length $\ell_c(x_{1:n})$ (in bits). Thus, the aim is to minimize the expected bits per sequence $L := E_{x \sim \rho}[\ell_c(x)]$, i.e., encoding rare sequences with more bits and frequent sequences with fewer bits. Shannon's source coding theorem establishes the limit on possible data compression as $L \geq H(\rho)$ for any possible code, where $H(\rho) := \mathbb{E}_{x \sim \rho}[-\log_2 \rho(x)]$ is the Shannon entropy (Shannon, 1948).

Arithmetic Coding Given a coding distribution ρ and a sequence $x_{1:n}$, arithmetic coding (Pasco, 1977; Rissanen, 1976) constructs a code with almost optimal length. It directly connects coding and compression with prediction and modeling: compressing well means modeling well in a log-loss sense and vice-versa. Assuming infinite precision for the arithmetic operations involved, the arithmetic code has length $-\lceil \log \rho(x_{1:n}) \rceil + 1$ bits, whereas the optimal code length is $-\log \rho(x_{1:n})$ bits. A practical implementation that is subject to B bit precision adds further $O(n 2^{-B})$ bits (Howard

& Vitter, 1991), which is negligible for 32- or 64-bit arithmetic. In the following we consider infinite precision arithmetic coders and refer to Witten et al. (1987) for the finite-precision implementation.

Arithmetic Encoder The arithmetic code of a sequence $x_{1:n}$ is the binary representation of a number $\lambda \in [0, 1)$. We identify λ by narrowing down an interval that encloses λ step by step (maintaining a growing prefix of the binary representation of λ throughout the process). Initially, this interval is $I_0 = [0, 1)$. In step $k > 0$ (i.e., encoding x_k), we first partition the previous interval $I_{k-1} = [l_{k-1}, u_{k-1})$ into N sub-intervals $\tilde{I}_k(x_1), \tilde{I}_k(x_2), \ldots$, one for each letter from $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$. The size of sub-interval $\tilde{I}_k(y)$ that represents letter y is $(u_{k-1} - l_{k-1}) \cdot \rho(y \mid x_{<k})$. Formally, we define

$$\tilde{I}_k(x) := \left[l_{k-1} + (u_{k-1} - l_{k-1}) \cdot \sum_{y<x} \rho(y \mid x_{<k}), \quad l_{k-1} + (u_{k-1} - l_{k-1}) \cdot \sum_{y \leq x} \rho(y \mid x_{<k}) \right), \tag{1}$$

assuming a strict order on $\mathcal{X}$. To encode x_k we proceed with its corresponding interval, i.e., $I_k = \tilde{I}_k(x_k)$. Finally, we choose $\lambda \in I_n$ with the shortest binary representation in the terminating interval I_n and use that binary representation to encode $x_{1:n}$. Fig. 1 illustrates this process.

Arithmetic Decoder Given λ and ρ decoding the k-th letter is easy: Starting with $I_0 = [0, 1)$, find y such that $\lambda \in \tilde{I}_k(y)$ to decode $x_k = y$, then set $I_k = \tilde{I}_k(x_k)$ and proceed with the k+1-st letter.

Likelihood Maximization In practice, the source distribution ρ is usually unknown and is instead estimated with a parametric probabilistic model $\hat{\rho}$. Thus, instead of achieving code length $-\sum_{i=1}^{n} \log_2 \rho(x_i \mid x_{<i})$ for the sequence $x_{1:n}$, we obtain the suboptimal length $-\sum_{i=1}^{n} \log_2 \hat{\rho}(x_i \mid x_{<i})$. As a result, the expected (suboptimal) number of bits is the *cross-entropy*:

$$H(\rho, \hat{\rho}) := \mathbb{E}_{x \sim \rho} \left[\sum_{i=1}^{n} -\log_2 \hat{\rho}(x_i \mid x_{<i}) \right]. \tag{2}$$

Thus, we can minimize the expected length of the encoded data stream with symbols distributed according to ρ by minimizing the cross-entropy with respect to some $\hat{\rho}$, which is equivalent to likelihood maximization (MacKay, 2003). However, Eq. (2) is exactly the same objective used to train current foundation models, i.e., the log-loss. Thus, minimizing the log-loss is equivalent to minimizing the compression rate of that model used as a lossless compressor with arithmetic coding, i.e., current language model training protocols use a maximum-compression objective.

Compression-Based Sequence Prediction Analogous to how a predictive distribution can be used for lossless compression via arithmetic coding (described above), any compressor can be employed for sequence prediction (Frank et al., 2000). The main idea is to define $\rho(x_{1:n})$ as the coding distribution $2^{-\ell_c(\cdot)}$, where $\ell_c(x_{1:n})$ is the length of sequence $x_{1:n}$ when encoded with compressor c (e.g., gzip). We thus recover the conditional distribution $\rho(x_i \mid x_{<i})$ by computing $2^{\ell_c(x_{<i}) - \ell_c(x_{<i}x_i)}$, for all x_i.

Universal Coding Above we discussed optimal (arithmetic) coding with respect to data sampled from a fixed distribution ρ. In contrast, universal (optimal) source coding with respect to all computable sampling distributions can, in theory, be achieved by choosing $\ell_c(x_{1:n})$ as the Kolmogorov complexity of $x_{1:n}$ (Kolmogorov, 1998; Li & Vitányi, 2019). For this choice, the conditional distribution described above is universally optimal over $x_{<i}$, recovering the Solomonoff predictor (Rathmanner & Hutter, 2011; Solomonoff, 1964a,b). The Solomonoff predictor is a Bayesian mixture of *all* predictors that can

be programmed in a chosen Turing-complete programming language. More precisely, for a predictor q of program-length $\ell_c(q)$ bits, the Solomonoff predictor assigns a prior weight of $2^{-\ell_c(q)}$ to predictor q. That is, if Q is the set of all predictors that can be programmed and computed, the Solomonoff predictor assigns probability $S(x_{1:n}) = \sum_{q \in Q} 2^{-\ell_c(q)} q(x_{1:n})$ to a sequence $x_{1:n}$, if every predictor q assigns that sequence probability $q(x_{1:n})$. Therefore, $S(x_{1:n}) \geq 2^{-\ell_c(q)} q(x_{1:n})$ for all $q \in Q$, and thus $-\log_2 S(x_{1:n}) \leq -\log_2 q(x_{1:n}) + \ell_c(q)$. Observe that $\ell_c(q)$ is a constant of q that is independent of the sequence length. Therefore, compressing optimally is equivalent to predicting optimally and vice versa (Hutter, 2005).

3. Experimental Evaluation

We now present our evaluation of the (in-context) compression capabilities of foundation models.

Compressors We compare our arithmetic coding-based language model compressors to two competitive general-purpose lossless compressors: gzip (Deutsch, 1996) and its improvement LZMA2 (Pavlov, 2019), used by the 7zip software. Both are based on Huffman coding (Huffman, 1952) and the Lempel-Ziv-Welch algorithm (Welch, 1984). We also consider specialized lossless compressors for image and audio data, i.e., PNG (Boutell, 1997) and FLAC (Coalson, 2008), respectively. Finally, we evaluate two types of language models (of different sizes) with arithmetic coding: vanilla decoder-only Transformers (Vaswani et al., 2017), which we pretrain on the enwik8 dataset, and pretrained Chinchilla-like foundation models (Hoffmann et al., 2022).

3.1. Datasets

We consider datasets of three different modalities, text, image, and audio, which have (a priori) very different biases for compression and thus provide a good testbed for evaluating a compressor's general capabilities. To render the results comparable across modalities, all our datasets are 1GB.

A key question is how to reconcile the different context lengths C of the compressors we consider. Transformers are restricted to short contexts ($C = 2048$ bytes, i.e., 2048 tokens of 8 bits that represent the ASCII characters, for our trained models and roughly 10 kilobytes for Chinchilla models), while gzip uses a maximum context of 32 kilobytes, and LZMA2 has a virtually "infinite" context length. Having a longer context allows a compressor to exploit more sequential dependencies to achieve a better compression rate. For compressors with finite contexts, there are two approaches to compress sequences that are longer than the context length: (i) slide the compressor byte by byte, thus always processing a history of the previous $C - 1$ bytes when compressing a new byte, and (ii) chunk the data stream into S sequences of C bytes and evaluate the in-context compression (without any history) averaged across batches. For Transformers, we consider the latter approach since sliding would increase their (already very long) running time by a factor of S. Therefore, we chunk all datasets into sequences of 2048 bytes and feed them to the compressors one-by-one. However, since classical compressors usually include a header in their compressed output, which can be larger than the compressed data in some cases, we only count it once for all batches, yielding a compression rate of $(\text{header} + \sum(l_c(\text{batch}) - \text{header}))/\text{num\_batches}$. Moreover, since chunking deteriorates the performance of classical compressors, which have context lengths $C \gg 2048$, we also report their compression rates on the unchunked datasets. We consider the following datasets:

enwik9 The enwik9 dataset (Hutter, 2006) consists of the first 1 000 000 000 (1 billion) bytes of the English Wikipedia XML dump on March 3rd, 2006 and is typically used to measure a model's

Table 1 | Compression rates (compressed size / raw size) on different datatsets (lower is better). The raw compression rate does not take the parameter size into account for the Transformer and Chinchilla models, while the adjusted compression rate considers the parameter size part of the compressed size. All datasets are of raw size 1GB. Random data is used as a baseline and should not be compressible. Transformer and Chinchilla are predictive models, which we use with arithmetic coding to obtain lossless compressors. We train the Transformer models from scratch on enwik8, while the Chinchilla models are pretrained on large text datasets. Transformers trained on enwik overfit to that data modality, while Chinchilla models are good compressors for various data types.

		Raw Compression Rate (%)				Adjusted Compression Rate (%)			
Chunk Size	**Compressor**	**enwik9**	**ImageNet**	**LibriSpeech**	**Random**	**enwik9**	**ImageNet**	**LibriSpeech**	**Random**
∞	gzip	32.3	70.7	36.4	100.0	32.3	70.7	36.4	100.0
	LZMA2	23.0	57.9	29.9	100.0	23.0	**57.9**	29.9	100.0
	PNG	42.9	58.5	32.2	100.0	42.9	58.5	32.2	100.0
	FLAC	89.5	61.9	30.9	107.8	89.5	61.9	30.9	107.8
2048	gzip	48.1	68.6	38.5	100.1	48.1	68.6	38.5	100.1
	LZMA2	50.0	62.4	38.2	100.0	50.0	62.4	38.2	100.0
	PNG	80.6	61.7	37.6	103.2	80.6	61.7	37.6	103.2
	FLAC	88.9	60.9	30.3	107.2	88.9	60.9	**30.3**	107.2
	Transformer 200K	30.9	194.0	146.6	195.5	30.9	194.0	146.6	195.5
	Transformer 800K	21.7	185.1	131.1	200.1	21.9	185.3	131.3	200.3
	Transformer 3.2M	17.0	215.8	228.2	224.0	**17.7**	216.5	228.9	224.7
	Chinchilla 1B	11.3	62.2	24.9	108.8	211.3	262.2	224.9	308.8
	Chinchilla 7B	10.2	54.7	23.6	101.6	1410.2	1454.7	1423.6	1501.6
	Chinchilla 70B	**8.3**	**48.0**	**21.0**	100.8	14008.3	14048.0	14021.0	14100.8

ability to compress data. It is an extension of the enwik8 dataset that only contains the first 100 million bytes. We train our vanilla Transformer models on enwik8, but evaluate on both enwik8 and enwik9 (to evaluate the out-of-distribution compression performance). While enwik8 is included in enwik9, it only represents the first 10% and thus still constitutes a significant distribution shift.

ImageNet The ImageNet dataset (Russakovsky et al., 2015) contains 14 197 122 annotated images from the WordNet hierarchy. Since 2010, the dataset has been used in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a benchmark in image classification and object detection. We extract contiguous patches of size 32×64 from all images, flatten them, convert them to grayscale (so that each byte represents exactly one pixel) to obtain samples of 2048 bytes. We then concatenate 488 821 of these patches, following the original dataset order, to create a dataset of 1 GB.

LibriSpeech LibriSpeech (Panayotov et al., 2015) is a corpus of approximately 1000 hours of 16kHz English speech. The data is derived from audiobooks from the LibriVox project and has been carefully segmented and aligned. We chunk the samples into batches of 2048 bytes and gather 488 821 such chunks into dataset of size 1 GB.

3.2. Comparing Compression Rates

Table 1 shows the compression rates for all compressors and datasets. We show both the raw compression rate, which does not take the model size (in bytes) into account, as well as the adjusted rate, which does. The size of the Python program for classical compressors is very small (a few kilobytes at most) and thus barely affects the compression rate. In contrast, language models suffer a

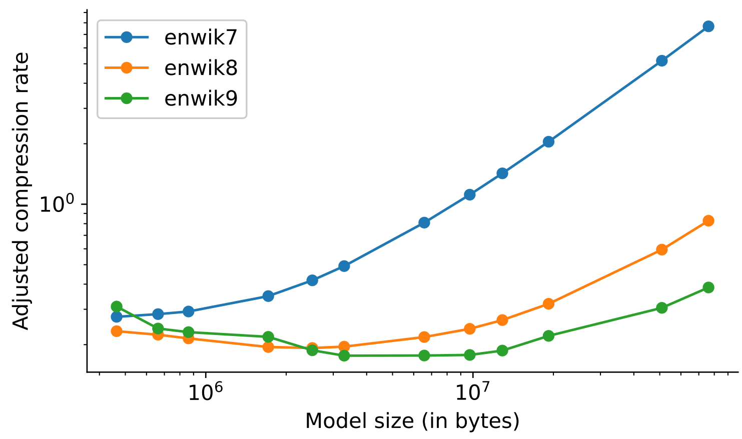


Figure 2 | Adjusted compression rates (compressed size / raw size) for Transformers of different sizes, trained on enwik8 and evaluated on enwik (both axes are logarithmic). Here, the compressed size does not only consider the size of the compressed output (roughly equal to the log-loss) but also the model size, which causes all curves to increase at some point. Every dataset gives rise to an optimal model size, with a good trade-off between performance (the size of the compressed data) and cost of the model (the number of parameters). The larger the dataset, the more parameters we can afford.

huge loss in compression rate due to their large size, which cannot be offset when compressing only 1GB of data. We encode each neural network parameter with 2 bytes, using a float16 representation since quantizing weights to this level does not significantly affect performance (Tao et al., 2022) and is standard for model inference. Note that further compressing the float16 parameters using classical compressors does not significantly reduce their size (we obtained rates of 92.2% and 89.1% on a 38M parameter Transformer with gzip and LZMA2, respectively). Also, recall that we only consider the offline setting, which computes the adjusted compression rate using a two-part code (i.e., it adds the model size to the log-loss of the data). In contrast, prequential (online) coding would provide an alternative view on adjusted compression by computing the adjusted compression rate as the log-loss plus the size of the training script (not the model parameters). According to prior work, prequential coding leads to better compression with overparametrized neural networks (Blier & Ollivier, 2018), however, it requires training the model online (which reduces performance and cannot be performed with foundation models) both during encoding and decoding (which is very costly for our models).

Foundation Models Are General-Purpose Compressors A lossless compressor induces an injective function over bit sequences, meaning that we cannot compress all sequences equally well (by the pigeonhole principle). Consequently, in practice, compressors are often tailored to a particular setting, e.g., FLAC for audio or PNG for images, and thus fail to compress other data modalities well (see Table 1). In contrast, general-purpose compressors, such as gzip, offer good performance on a wide range of data sources. Surprisingly, Chinchilla models, while trained primarily on text, also appear to be general-purpose compressors, as they outperform all other compressors, even on image and audio data (see Table 1). Note that Chinchilla models have not been trained on this kind of data according to Appendix A. of Hoffmann et al. (2022), which states that the training dataset consists of a mix of internet text data (Wikipedia, websites, github) and books. However, it is still possible (but unlikely) that some images or audio samples were encoded into text on some websites. Thus, Chinchilla models achieve their impressive compression performance by conditioning a (meta-)trained model to a particular task at hand via in-context learning (Genewein et al., 2023). In contrast, smaller

Transformers, trained manually on enwik8, only achieve good compression rates on similar Wikipedia data, i.e., enwik9. However, larger models' stronger in-context compression (or in-context learning) comes at a price: the number of parameters, which has to be offset with increasingly large data sources when computing the adjusted compression rate (see Section 3.3). Finally, note that since Chinchilla has been trained on Wikipedia, the enwik9 results are in-distribution.

3.3. Optimal Model-Dataset Size Tradeoff

As shown in Table 1, foundation models incur a huge cost in compression rates when accounting for their size, which is in the order of hundreds of GBs for billions of parameters. In theory, if the dataset is infinite, we can ignore the model's size since it is insignificant compared to the size of the dataset. However, in practice, a foundation model can only achieve non-trivial (adjusted) compression rates when evaluated on datasets in the order of TBs (or more). Since this is infeasible under reasonable hardware constraints, we instead investigate the optimal model size with smaller Transformers that we train on enwik8. Recall that the model size (in bytes) is twice the number of (float16) parameters.

Fig. 2 visualizes the adjusted compression rate for vanilla Transformers of different model sizes for the enwik datasets. We observe that larger models achieve better compression rates on larger datasets, thus justifying recent trends in model scaling (Kaplan et al., 2020). However, they achieve worse rates on smaller datasets, indicating that scaling laws are, in fact, dependent on the size of the test set. That is, for each dataset, the model sizes reach a critical point, after which the adjusted compression rate starts to increase again since the number of parameters is too big compared to the size of the dataset. Note that we evaluate offline compression, i.e., we do not necessarily compress the data the model was trained on, meaning that the results on enwik7 and enwik8 are in-distribution, while the enwik9 results are out-of-distribution. Nevertheless, larger models still achieve better compression rates on enwik9 than enwik8, illustrating the benefits of scaling.

3.4. Compressors as Generative Models

In Section 2, we discussed how any compressor can be employed as a sequence prediction model. Concretely, for compressor c, we sample the next byte according to the distribution $\hat{\rho}(x_i \mid x_{<i}) \sim 2^{\ell_c(x_{<i})-\ell_c(x_{<i}x_i)}$, i.e., we compute the length ℓ_c of the compressed sequence $c(x_{<i}b)$ for all possible $b \in \mathcal{X}$. Thus, if a byte b leads to a particularly short compressed sequence (when concatenated with $x_{<i}$), it will have a higher probability of being sampled next. Note that any constant in the length function (e.g., the header for classical compressors) disappears when we normalize the distribution.

Since generic compressors have a low intrinsic bias, sampling data without conditioning does not yield interesting results as it looks random. Thus, we condition the compressors on part of an existing sequence (1948 bytes for enwik9, half of the sample for ImageNet and LibriSpeech) and generate the remaining bytes with the compression-based generative model. We compare the generative performance of gzip and Chinchilla 70B across all three data modalities in Figs. 3 to 5 for text, image, and audio data, respectively. In general, generative models can be evaluated using one of two ways: sampling the next byte $\hat{\rho}(x_i \mid x_{<i})$ (i) using teacher forcing, i.e., conditioning on the true subsequence $x_{<i}$, or (ii) via autoregressive sampling, i.e., conditioning on the model's previous outputs. The latter induces a distribution shift, and with it undesired side effects (Ortega et al., 2021), but is standard and thus what we choose to visualize.

Context Text (1948 Bytes)

ction Act 1876]]. They are selected by the Prime Minister, but are formally appointed by the Sovereign. A Lord of Appeal in Ordinary must retire at the age of 70, or, if his or her term is extended by the Government, at the age of 75; after reaching such an age, the Law Lord cannot hear any further legal cases. The number of Lords of Appeal in Ordinary (excluding those who are no longer able to hear cases due to age restrictions) is limited to twelve, but may be changed by [[statutory instrument]]. Lords of Appeal in Ordinary traditionally do not participate in political debates, so as to maintain judicial independence. Lords of Appeal in Ordinary hold seats the House of Lords for life, remaining members even after reaching the retirement age of 70 or 75. Former Lord Chancellors and holders of other high judicial office may also sit as Law Lords under the Appellate Jurisdiction Act, although in practice this right is infrequently exercised. After the coming into force of the Constitutional Reform Act 2005, the Lords of Appeal in Ordinary will become judges of the Supreme Court of the United Kingdom and will be barred from sitting or voting until they retire as judges.\n\nThe largest group of Lords Temporal, and indeed of the whole House, are [[Life peer|life peers]]. Life peers with seats in the House of Lords rank only as barons or baronesses, and are created under the [[Life Peerages Act 1958]]. Like all other peers, life peers are created by the Sovereign, who acts on the advice of the Prime Minister. By convention, however, the Prime Minister allows leaders of other parties to select some life peers so as to maintain a political balance in the House of Lords. Moreover, some non-party life peers (the number being determined by the Prime Minister) are nominated by an independent House of Lords Appointments Commission. If an hereditary peer also holds a life peerage, he or

Ground Truth (100 Bytes)

– she remains a member of the House of Lords without a need for an election. In [[2000]], the governm

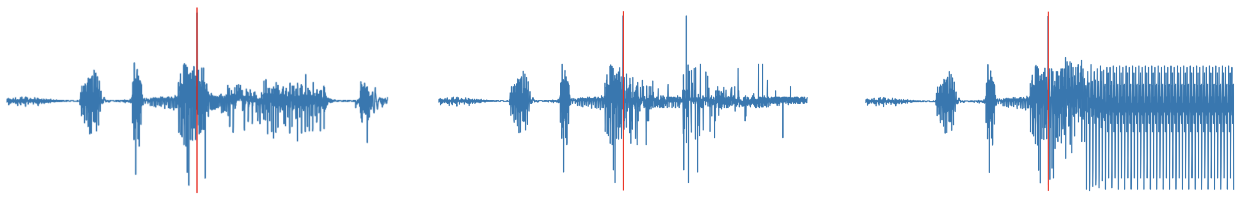
gzip Samples (100 Bytes)

– (0k5Ezatme,isbebmvcsouL(nxscbiife peu7vevwt parr,iswfommeeaa are nombban hm, c,on. , pncmm.sexg uam
– Suasa8g thformp0iufoof Lo e7vkoasaeka w8viiufoounb,xbepe,deto.,5mdrSu r,teepe,rgesgS,be.dcyh2vLnary
– CxOsic,*auEfOlnknm } eaa0oplutfpq(afcnuChanm,areovervr LoventiL.myehm;nrhvnywsaO7seeg Apo,arelyehm;.

Chinchilla 70B Samples (100 bytes)

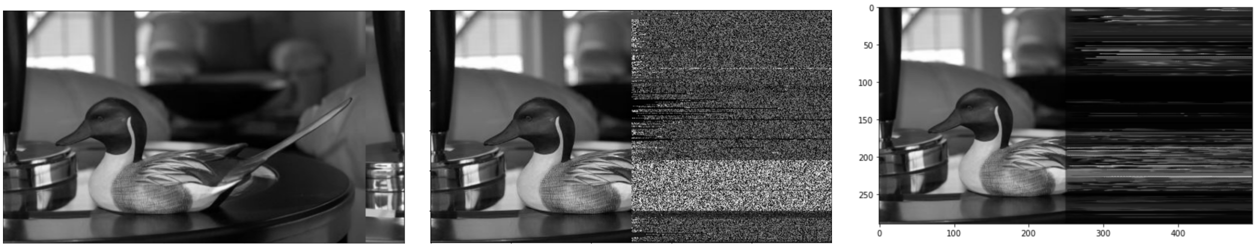
– she may use either title, but the hereditary peerage is considered to be superior. Lords Temporal c
– she may choose which title to use, though the title of the life peerage is normally used. The Sover
– she may elect to sit in the House as a life peer, rather than as a hereditary peer. Life peers are

Figure 3 | Compression-based generation for text data. We condition gzip and Chinchilla on a context text of size 1948 bytes (from enwik9) and then sample 100 bytes (N tokens) autoregressively. Since Chinchilla employs a tokenizer, the sampled sequences will contain N tokens, which do not necessarily decode to 100 bytes. Chinchilla's predictions are significantly more coherent than gzip's.



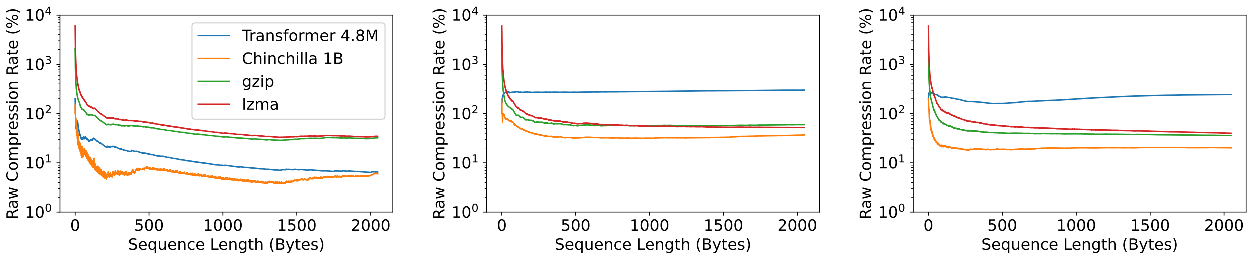
(a) Original spectrogram (b) gzip (c) Chinchilla

Figure 4 | Compression-based generation for audio data. We condition gzip and Chinchilla on the first 1024 bytes of the base sequence (from LibriSpeech) and then sample the remaining 1024 bytes autoregressively. Chinchilla predictions exhibit a typical "loop" pattern of autoregressive generation.



(a) Original image (b) gzip (row-wise) (c) Chinchilla (row-wise)

Figure 5 | Compression-based generation for image data. We condition gzip and Chinchilla on the first half of every row of the ImageNet image and then sample the remaining half autoregressively. Both models produce incoherent samples, but Chinchilla looks much less noisy than gzip.



(a) enwik9 (b) ImageNet (c) LibriSpeech

Figure 6 | In-context compression rate over sequence length. For every dataset, we compute the compression rate for all subsequences of 2048 bytes, averaged over 100 sequences.

3.5. Sequential Evolution of In-Context Compression

Language models take a very different "approach" to compression compared to classical compressors. Classical compressors have a small program size and optimize for a large context length to exploit sequential dependencies in the data. In contrast, foundation models consist of billions of parameters, which enable rapid adaptation in their (relatively) short context window (Genewein et al., 2023). Thus, arithmetic coding-based compressors rely heavily on the predictive models' in-context learning capabilities to achieve competitive compression performance. We investigate this phenomenon in Fig. 6, which visualizes the compression rate across sequence lengths for gzip, Chinchilla 1B and a Transformer pretrained on enwik8. Intuitively, the longer the sequence, the more data the model can process in its context, and therefore, the better the compression. As expected, most compression rates decrease quickly with increasing sequence length, indicating that the models learn some data statistics in-context, without any gradient-based training. As in Table 1, the Chinchilla model achieves the best compression rates accross all three data modalities and sequence lengths.

3.6. Tokenization Is Compression

Transformers are generally not trained on raw input data but on tokenized versions thereof, both for efficiency and performance reasons. As a consequence, Transformers are trained on compressed data, with tokenizers acting as the compressor. Since tokenization is known to have an impact on the generalization performance (Radford et al., 2019), we investigate its impact on the compression rate in Table 2. Concretely, we train Transformers on enwik8 using different tokenizers: ASCII, i.e., an alphabet of size 256 (no tokenization), and byte-pair encoding trained on enwik8, with various

Table 2 | Raw compression rates (compressed size / raw size) on enwik9 for Transformers trained on enwik8 with different tokenizers, ASCII and byte-pair encoding (BPE), with various vocabulary sizes. Transformers compress better with simpler tokenizers. However, larger vocabulary sizes reduce the length of the sequence more, meaning more information can be packed into the context.

	Raw Compression Rate (%)		
Tokenization	**200K**	**6.4M**	**38M**
ASCII	22.9	**13.6**	**6.4**
BPE 1000	25.4	14.8	6.9
BPE 2000	25.6	15.7	7.4
BPE 5000	23.1	17.1	8.7
BPE 10000	21.3	17.0	8.9
BPE 20000	**19.3**	16.4	9.0

vocabulary sizes (1K, 2K, 5K, 10K, and 20K tokens). Note that the tokenizations are lossless.

Increasing the number of tokens (i.e., the "alphabet size") reduces the length of the sequence and thus increases the amount of information in a models context. However, decreasing the sequence length comes at a price: the number of tokens is larger, which makes the prediction task more challenging since reducing the entropy of the conditional distribution $\rho(x_i \mid x_{<i})$ is increasingly difficult for larger alphabet size. In theory, as the tokenization is a lossless compression, the two effects should compensate. In practice, we observe that if the model is small, increasing the number of possible tokens boosts the compression performance. In contrast, for bigger models, it seems that the converse happens: having a larger token vocabulary harms the final compression rate of the model. Nevertheless, short sequence lengths also help Transformers since their time complexity scales quadratically with context length, and it has been shown they do not generalize well to long contexts (Delétang et al., 2023; Ruoss et al., 2023). This explains why most practical Transformer implementations still use some form of tokenization, e.g., SentencePiece (Kudo & Richardson, 2018).

4. Related work

Prediction vs. Compression Leveraging Shannon's source coding theorem (Shannon, 1948), a plethora of approaches exploit the connection between prediction and compression. For example, context-tree weighting (CTW) (Willems et al., 1995) mixes the predictions of many underlying Markov models to achieve lossless compression via arithmetic coding (Pasco, 1977; Rissanen, 1976). Similarly, prediction by partial matching (PPM) (Cleary & Witten, 1984) also leverages arithmetic coding, but uses a contiguous context matching method to create probability distributions based on the history of characters in a sequence. Likewise, PAQ8 (Knoll & de Freitas, 2012) uses a weighted combination of predictions from a large number of models (most of them based on context matching, but unlike PPM also noncontiguous context matches). In a different setting, Veness et al. (2015) demonstrated how to employ compression to obtain value estimates of a policy in an environment. Frank et al. (2000) and later Teahan & Harper (2003) introduced the idea of classification with compressors. Recently, Jiang et al. (2023) applied this technique with NLP tasks, paired with a k-nearest-neighbour algorithm. The results are surprisingly good for simple general purpose compressors like gzip. Jiang et al. (2022) exploit the same idea but train the compressor on a vast amount of unlabeled data first. Finally, van den Oord & Schrauwen (2014) apply arithmetic coding to image compression using Student distribution mixtures and Gaussian processes as predictors.

Compression With Neural Networks Prior work demonstrated that neural predictive distributions can be employed to perform lossless compression via arithmetic coding (Cox, 2016; Goyal et al., 2019; Knoll, 2014; Liu et al., 2019; Mahoney, 2000; Mentzer et al., 2019, 2020; Mikolov, 2012; Rhee et al., 2022; Schiopu & Munteanu, 2020; Schiopu et al., 2018; Schmidhuber & Heil, 1996). Similarly, neural networks were also shown to achieve strong lossless compression rates when replacing arithmetic coding with asymmetric numeral systems (Barzen et al., 2022; Hoogeboom et al., 2019; Kingma et al., 2019; Townsend et al., 2019). While these approaches assume the existence of a separate training set, a different line of work investigated arithmetic coding-based neural compression in a purely online fashion, i.e., training the model only on the data stream that is to be compressed (Bellard, 2019, 2021; Goyal et al., 2020; Mao et al., 2022). Finally, concurrent work (Valmeekam et al., 2023) also investigated lossless offline compression with foundation models, using arithmetic coding with LLaMA-7B (Touvron et al., 2023).

Compression Biases: Tokenization, Model Size, etc. Much effort has been devoted on understanding the inductive biases of neural networks. Here, we are mostly interested in the biases of Natural Language Processing (NLP) and Transformers. Kudo & Richardson (2018) defined a tokenizer for NLP-related research, an improvement of well-known techniques like byte-pair encoding (BPE) (Sennrich et al., 2016), BPE dropout (Provilkov et al., 2020), and subword regularization (Kudo, 2018). In this paper, we show how these tokenization techniques act as pre-compressors for the data, and can significantly affect the final compression rates when paired with a neural model. More general studies have been performed on generalization (Neyshabur et al., 2017), which, we argue, is equivalent to the model's compressive power when accounting parameters code-length. Finally, some work has been done on compressing the neural models' parameters themselves (Cheng et al., 2017).

5. Conclusion

In this paper we investigated how and why compression and prediction are equivalent. Arithmetic coding transforms a prediction model into a compressor, and, conversely, a compressor can be transformed into a predictor by using the coding lengths to construct probability distributions following Shannon's entropy principle. We evaluated large pretrained models used as compressors against various standard compressors, and showed they are competitive not only on text but also on modalities they have never been trained on (images, audio data). We showed that the compression viewpoint provides novel insights on scaling laws since it takes the model size into account, unlike the log-loss objective, which is standard in current language modeling research. Consequently, we showed that the optimal model size is inextricably linked to the dataset size and cannot be scaled without limit.

Acknowledgments

We thank Jörg Bornschein, Nando de Freitas, Slav Petrov, and Zhengdong Wang for their helpful feedback and insightful discussions.

References

Benjamin Lukas Cajus Barzen, Fedor Glazov, Jonas Geistert, and Thomas Sikora. Accelerated deep lossless image coding with unified paralleleized GPU coding architecture. In *PCS*, 2022.

Fabrice Bellard. Lossless data compression with neural networks. Technical report, Amarisoft, 2019.

Fabrice Bellard. NNCP v2: Lossless data compression with transformer. Technical report, Amarisoft, 2021.

Léonard Blier and Yann Ollivier. The description length of deep learning models. In *NeurIPS*, 2018.

Rishi Bommasani et al. On the opportunities and risks of foundation models. *arXiv:2108.07258*, 2021.

Thomas Boutell. PNG (portable network graphics) specification version 1.0. *RFC*, 1997.

Tom B. Brown, Benjamin Mannand Nick Ryder, Melanie Subbiah, et al. Language models are few-shot learners. In *NeurIPS*, 2020.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott M. Lundberg, Harsha Nori, Hamid Palangi, Marco Túlio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv:2303.12712*, 2023.

Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. Scaling transformer to 1m tokens and beyond with RMT. *arXiv:2304.11062*, 2023.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv:1710.09282*, 2017.

John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Trans. Commun.*, 1984.

Josh Coalson. Free lossless audio codec, 2008. URL https://xiph.org/flac.

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

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. Neural networks and the chomsky hierarchy. In *ICLR*, 2023.

Peter Deutsch. GZIP file format specification version 4.3. *RFC*, 1996.

Jarek Duda. Asymmetric numeral systems. *arXiv:0902.0271*, 2009.

Eibe Frank, Chang Chui, and Ian H. Witten. Text categorization using compression models. In *Data Compression Conference*, 2000.

Tim Genewein, Grégoire Delétang, Anian Ruoss, Li Kevin Wenliang, Elliot Catt, Vincent Dutordoir, Jordi Grau-Moya, Laurent Orseau, Marcus Hutter, and Joel Veness. Memory-based meta-learning on non-stationary distributions. *arXiv:2302.03067*, 2023.

Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. Deepzip: Lossless data compression using recurrent neural networks. In *DCC*, 2019.

Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. Dzip: Improved general-purpose lossless compression based on novel neural network modeling. In *DCC*, 2020.

Mandy Guo, Joshua Ainslie, David C. Uthus, Santiago Ontañón, Jianmo Ni, Yun-Hsuan Sung, and Yinfei Yang. Longt5: Efficient text-to-text transformer for long sequences. In *NAACL-HLT (Findings)*, 2022.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, et al. Training compute-optimal large language models. *arXiv:2203.15556*, 2022.

Emiel Hoogeboom, Jorn W. T. Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression. In *NeurIPS*, 2019.

Paul G. Howard and Jeffrey Scott Vitter. Analysis of arithmetic coding for data compression. In *Data Compression Conference*, 1991.

David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 1952.

Marcus Hutter. *Universal Artificial Intellegence - Sequential Decisions Based on Algorithmic Probability*. Springer, 2005.

Marcus Hutter. 500'000€ prize for compressing human knowledge, 2006. URL http://prize.hutter1.net.

Zhiying Jiang, Yiqin Dai, Ji Xin, Ming Li, and Jimmy Lin. Few-shot non-parametric learning with deep latent variable model. In *NeurIPS*, 2022.

Zhiying Jiang, Matthew Y. R. Yang, Mikhail Tsirlin, Raphael Tang, Yiqin Dai, and Jimmy Lin. "low-resource" text classification: A parameter-free classification method with compressors. In *ACL (Findings)*, 2023.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.

Friso H. Kingma, Pieter Abbeel, and Jonathan Ho. Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. In *ICML*, 2019.

Byron Knoll. CMIX, 2014. URL http://www.byronknoll.com/cmix.html.

Byron Knoll and Nando de Freitas. A machine learning perspective on predictive coding with PAQ8. In *DCC*, 2012.

Andrei N. Kolmogorov. On tables of random numbers. *Theoretical Computer Science*, 1998.

Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In *ACL (1)*, 2018.

Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *EMNLP (Demonstration)*, 2018.

Michael Laskin, Luyu Wang, et al. In-context reinforcement learning with algorithm distillation. In *ICLR*. OpenReview.net, 2023.

Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, 4th Edition*. Springer, 2019.

Qian Liu, Yiling Xu, and Zhu Li. DecMac: A deep context model for high efficiency arithmetic coding. In *ICAIIC*, 2019.

David J. C. MacKay. *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.

Matthew V. Mahoney. Fast text compression with neural networks. In *FLAIRS*, 2000.

Yu Mao, Yufei Cui, Tei-Wei Kuo, and Chun Jason Xue. TRACE: A fast transformer-based general-purpose lossless compressor. In *WWW*, 2022.

Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Practical full resolution learned lossless image compression. In *CVPR*, 2019.

Fabian Mentzer, Luc Van Gool, and Michael Tschannen. Learning better lossless compression using lossy compression. In *CVPR*, 2020.

Tomas Mikolov. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno Universtiy of Technology, 2012.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *NIPS*, 2017.

Pedro A. Ortega, Markus Kunesch, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Joel Veness, Jonas Buchli, Jonas Degrave, Bilal Piot, Julien Pérolat, Tom Everitt, Corentin Tallec, Emilio Parisotto, Tom Erez, Yutian Chen, Scott E. Reed, Marcus Hutter, Nando de Freitas, and Shane Legg. Shaking the foundations: delusions in sequence models for interaction and control. *arXiv:2110.10819*, 2021.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An ASR corpus based on public domain audio books. In *ICASSP*, 2015.

Richard C. Pasco. Source coding algorithms for fast data compression (ph.d. thesis abstr.). *IEEE Trans. Inf. Theory*, 1977.

Igor Pavlov. 7z Format, 2019. URL http://www.7-zip.org/7z.html.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. Bpe-dropout: Simple and effective subword regularization. In *ACL*, 2020.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Jack W. Rae et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv:2112.11446*, 2021.

Samuel Rathmanner and Marcus Hutter. A philosophical treatise of universal induction. *Entropy*, 2011.

Hochang Rhee, Yeong Il Jang, Seyun Kim, and Nam Ik Cho. LC-FDNet: Learned lossless image compression with frequency decomposition network. In *CVPR*, 2022.

Jorma Rissanen. Generalized kraft inequality and arithmetic coding. *IBM J. Res. Dev.*, 1976.

Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bennani, Shane Legg, and Joel Veness. Randomized positional encodings boost length generalization of transformers. In *ACL (2)*, 2023.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 2015.

Ionut Schiopu and Adrian Munteanu. Deep-learning-based lossless image coding. *IEEE Trans. Circuits Syst. Video Technol.*, 2020.

Ionut Schiopu, Yu Liu, and Adrian Munteanu. CNN-based prediction for lossless coding of photographic images. In *PCS*, 2018.

Jürgen Schmidhuber and Stefan Heil. Sequential neural text compression. *IEEE Trans. Neural Networks*, 1996.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL (1)*, 2016.

Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 1948.

Ray J. Solomonoff. A formal theory of inductive inference. part I. *Inf. Control.*, 1964a.

Ray J. Solomonoff. A formal theory of inductive inference. part II. *Inf. Control.*, 1964b.

Chaofan Tao, Lu Hou, Wei Zhang, Lifeng Shang, Xin Jiang, Qun Liu, Ping Luo, and Ngai Wong. Compression of generative pre-trained language models via quantization. In *ACL (1)*, 2022.

William J. Teahan and David J. Harper. *Using Compression-Based Language Models for Text Categorization*, pp. 141–165. Springer Netherlands, 2003.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. Llama: Open and efficient foundation language models. *arXiv:2302.13971*, 2023.

James Townsend, Thomas Bird, and David Barber. Practical lossless compression with latent variables using bits back coding. In *ICLR (Poster)*, 2019.

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

Aäron van den Oord and Benjamin Schrauwen. The student-t mixture as a natural image patch prior with application to image compression. *J. Mach. Learn. Res.*, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

Joel Veness, Marc G. Bellemare, Marcus Hutter, Alvin Chua, and Guillaume Desjardins. Compress and control. In *AAAI*, 2015.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Terry A. Welch. A technique for high-performance data compression. *Computer*, 1984.

Frans M. J. Willems, Yuri M. Shtarkov, and Tjalling J. Tjalkens. The context-tree weighting method: basic properties. *IEEE Trans. Inf. Theory*, 1995.

Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Commun. ACM*, 1987.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *NeurIPS*, 2020.