Cuaċ: Fast and Small Universal Representations of Corpora

John P. McCrae[†,‡], **Bernardo Stearns**[†], **Alamgir Munir Qazi**,
Shubhanker Banerjee[‡] and **Atul Kr. Ojha**[†]
[†]Research Ireland Insight Centre
[‡]Research Ireland ADAPT Centre
University of Galway
Ireland
john@mccr.ae

Abstract

The increasing size and diversity of corpora in natural language processing requires highly efficient processing frameworks. Building on the universal corpus format, Teanga, we present Cuaċ, a format for the compact representation of corpora. We describe this methodology based on short-string compression and indexing techniques and show that the files created with this methodology are similar to compressed human-readable serializations and can be further compressed using lossless compression. We also show that this introduces no computational penalty on the time to process files. This methodology aims to speed up natural language processing pipelines and is the basis for a fast database system for corpora.

1 Introduction

The size and scope of corpora used in natural language processing (NLP) applications have grown massively in the past few years and as such the efficient storage and retrieval of large-scale linguistic corpora are critical for these applications. The growth of textual data means the traditional storage and annotation formats can present significant challenges to real-world applications. We recently proposed the Teanga format (McCrae et al., 2024), which provides a universal method of annotating corpora, principally through a metamodel serialized in YAML and other formats. While this model addresses key challenges in the accessibility and interoperability of corpora, the YAML format does not provide an efficient method for working with corpora in this data model. In this paper, we present Cuaċ[1] (Compression of Universal Annotated Corpora), a new serialization method, which addresses these challenges by providing a compact, high-performance representation of annotated corpora.

[1]Cuaċ (/kuəx/, 'cuach' in standard orthography) means 'cuckoo' and 'bundle' in Irish.

Cuaċ is designed to reduce the size of corpora as stored on disk, while still allowing full searchability of the corpus and to avoid increasing processing times. It is built on top of the Teanga data model, and integrates annotation layers using multiple compression techniques in order to minimize the redundant representation of information. It also incorporates lightweight text compression methods, alongside indexed integer representations to reduce the storage size of the data.

In this paper, we present the Cuaċ format and its implementation within the Teanga framework. We discuss the compression strategies and the indexing mechanisms. We evaluate its performance in terms of the file size reduction as well as the time taken to process the records. Our results show that Cuaċ not only outperforms conventional formats such as YAML and JSON, which are similar to the XML and CoNLL-U formats used originally for the corpora, but also strongly outperforms Parquet, a binary format that is widely used for sharing datasets including corpora. As such Cuaċ is a practical solution for handling large-scale annotated corpora in NLP research and applications.

2 Related Work

Effective compression and representation of large data sets are essential for scalable storage and retrieval. Brotli (Alakuijala et al., 2015), a compression algorithm developed by Google, provides high compression ratios through a static dictionary and transformation techniques, making it ideal for web and textual data compression. In contrast to traditional methods like Deflate, Brotli greatly enhances both compression density and decompression speed, positioning it as a strong candidate for compact corpus representation.

When dealing with structured data, especially RDF (Resource Description Framework), compression strategies concentrate on removing structural redundancies. The HDT (Header-Dictionary-

Triples) format (Fernández et al., 2013) is a commonly used binary serialization that streamlines RDF data by replacing textual terms with numeric identifiers (Hernández-Illera et al., 2020). Advanced optimizations like HDT++ build on this by utilizing schema-based redundancies, such as families of predicates and typed subjects, effectively reducing storage needs by half compared to standard HDT serialization (Hernandez-Illera et al., 2015).

Another innovative method, k²-triples, enhances RDF graph compression by dividing RDF triples into predicate-specific binary matrices, which are then encoded using k²-trees. This strategy improves structural compression while ensuring efficient query performance. Furthermore, RDF-Tr enhances RDF compression by reorganizing triples to reflect recurring structural patterns, which leads to better space efficiency and faster retrieval speeds (Hernández-Illera et al., 2020).

Advancements in RDF data compression have led to the development of grammar-based techniques, such as gRDF, which utilizes the gRePair algorithm to identify and compress repetitive patterns within RDF datasets, achieving substantial reductions in data size while preserving structural integrity (Sultana and Lee, 2022). Additionally, compressed indexing methods, including trie-based layouts and circular suffix sorting, have been introduced to compactly represent RDF triples, enabling efficient pattern-matching operations and enhancing query execution speeds (Perego et al., 2021; Brisaboa et al., 2023). These innovations address the challenges posed by the increasing volume of RDF data, facilitating more efficient storage and retrieval processes.

Beyond grammar-based and indexing techniques, researchers have explored estimation-based optimizations for compressing RDF knowledge bases. These methods analyze input and intermediate data to improve compression efficiency, reducing storage overhead while preserving query performance (Wang et al., 2024). Additionally, machine learning-driven compression has gained traction, with inductive autoencoders learning compact representations of RDF graphs by identifying latent structures and redundant patterns, leading to improved storage efficiency and faster retrieval (Sultana et al., 2024). Such techniques signal a shift toward hybrid approaches that integrate statistical learning with structural compression, paving the way for more scalable RDF management solutions. These advancements in compression and serialization demonstrate the potential for fast and small universal representations of corpora, balancing storage efficiency with rapid access and processing capabilities.

3 Methodology

3.1 Teanga Data Model

Teanga (McCrae et al., 2024) is a framework to represent and share annotated linguistic corpora. By offering a simple, flexible, and interoperable format for natural language processing (NLP) tasks, it makes linguistic corpus FAIR (Findable, Accessible, Interoperable, and Reusable; see Wilkinson et al. (2016)). It handles common problems found in linguistic data pipelines such as lack of standardization, verbosity of linked data models, and destructive annotation in formats like TEI (Ide and Sperberg-McQueen, 1995) and CoNLL-X (Buchholz and Marsi, 2006). Teanga is based on a layered annotation approach, with multiple types of layers available. A base character layer for raw text and annotation layers (span, division, element, sequence) for linguistic information. All annotations are implemented as stand-off layers and may refer directly to the character layer by character offsets or may refer to another annotation layer, for example, a part-of-speech layer may reference a token layer instead and this can be mapped onto the character layer by means of the offsets in the token layer. Further, to make the framework more flexible offsets can be given in four different ways:

Span A start and (exclusive) end index are given for each annotation, e.g., named entities.

Element Only a start index is given, the end index is assumed to be one element later, e.g., misspelled words.

Division Only a start index is given, the end index is assumed to be the same as the start index of the next annotation, e.g., sentences or paragraphs.

Sequence No indexes are given. The annotations must exactly follow in a one-to-one correspondence with the base layer, e.g., part-of-speech tags.

In Figure 1, we see some examples of these layers. First, we have a text layer giving the characters

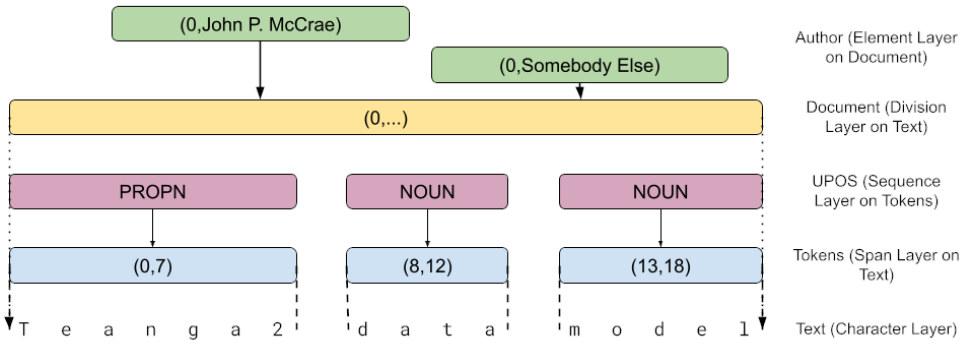

Figure 1: Example of Teanga Layer Types

and the token layer giving the tokens by character offsets. A part of speech layer is then given as a sequence layer based on the token layer, i.e., there is one part-of-speech tag for each token. Then, the text can divided into multiple layers such as paragraphs, chapters or documents with a division layer and finally, annotations on the document such as authors can be given in the element layer[2].

Teanga supports YAML and JSON serialization, which can be converted to RDF for linked data integration. Teanga can be used for a wide range of annotations syntactic, sentence and word alignments, multilingual corpora, and other NLP tasks. Teanga is released as a Python library that interfaces to a Rust code base to enable large-scale processing, efficiently storing and querying massive corpora on disk.

To expand Teanga's efficiency and scalability, we integrate compression frameworks into Teanga to optimize the storage and processing of annotated corpora. Ensuring that Teanga remains not only FAIR-compliant but also highly performant, even for massive datasets.

3.2 Text Compression

Traditional compression algorithms such as GZIP (Deutsch, 1996b), DEFLATE (Deutsch, 1996a), or XZ (Collin, 2010) perform poorly on very short strings, often producing output larger than the input. This inefficiency stems from the overhead of dictionary building and metadata storage, which becomes proportionally significant for small inputs. To address this specific challenge, specialized compression libraries for short strings have been developed, with SMAZ (Sanfilippo, 2009) and SHOCO (Schramm, 2014) being two notable examples.

SMAZ (Sanfilippo, 2009) employs a fixed-codebook compression strategy with 254 entries containing common character sequences derived from English text and web content. The algorithm iteratively scans input strings to identify the longest possible substring present in its codebook, replacing matches with corresponding single-byte codes. Non-matching content is encoded verbatim using designated marker bytes: 254 for individual characters and 255 for character sequences.

This approach demonstrates efficacy for strings as short as 2-3 bytes, achieving compression rates of 40-50% for English text and structured content such as URLs. The implementation prioritizes simplicity and computational efficiency, comprising approximately 200 lines of C code. However, the static nature of its codebook limits SMAZ's effectiveness when processing numerical data, non-English text, or domain-specific content that diverges from its optimization target.

SHOCO (Schramm, 2014) implements a statistical compression methodology based on character frequency distributions and bigram analysis. The algorithm exploits the unused most significant bit in ASCII characters (which is always zero) to differentiate between compressed and literal encoding modes. For compression, SHOCO utilizes a character-successor model where encoding efficiency is determined by positional context and statistical frequency. SHOCO's statistical approach enhances flexibility through customizable models, enabling adaptation to diverse data types. However, this reliance on a statistical model introduces higher computational overhead and memory requirements compared to SMAZ's simpler fixed-codebook approach.

[2]Note that annotations may alternatively be given as metadata fields

The Cuaċ format supports both of these compression methods. SMAZ uses hard-coded values that are optimal for English and are used as a default. For other languages, we offer the choice of either using the default English-optimized model or a new trained model stored in the header of the Cuaċ file. This model is trained on a subsection at the start of the corpus which is at least a certain number of bytes (default value is 10^6 bytes) or all the data if the whole file does not match this limit. This mode is called *generate* mode and involves serializing the compression model within the data file, in contrast, the standard SMAZ and SHOCO models use tables stored in the compressor executable file.

3.3 Indexes

Apart from textual data, the rest of the data within the Teanga model can be represented as lists of integers. The major forms of this include:

Offsets Annotations in element, division or span layers consist of an integer pointing to the index in the base layer. For example, a token layer is a span layer that gives an index to the character layer (its base layer) by means of character offsets. As such, the offsets consist of one list of integers or two lists of integers for span layers. For span layers, these values are not interpolated, e.g., the lists of all start indexes are stored first and then the list of all end indexes in order.

Link The data contained in a layer may be links to other annotations, e.g., for parse trees or annotations. This is naturally a list of integers.

Enum The data may be a value from a fixed list given in the metadata. In this case, this is converted to a list by means of using the index of the annotation in the metadata.

In all cases, these lists of integers are stored in the following way. Firstly the first list of integers is checked as to whether it is strictly ascending in values. If this is the case, the list is transformed into a delta where each value is stored as the difference to the previous value. The second list of integers is then checked to see if all of its values are greater than the corresponding index in the first index. If so, the second list is transformed by taking the delta to the first list. If there is a third list of integers, which must be link data, then this is not altered. For example, the tokenization of "I love Teanga a lot", which is written in JSON as `[0,1],[2,6],[7,12],[13,14],[15,18]` will be transformed to:

```
[0,2,5,6,2][1,4,5,1,3]
```

The goal of this transformation is to ensure that the numbers used in the list are small non-negative integers. We then calculate the single largest value in each list and the number of bits required to store it. The data is then stored in *variable-precision* format where a single byte first gives the precision in bits and then each other number is stored in order. So for the example above, both lists can be stored in 3-bit precision so a total of 6 bytes are required, 1 byte for each precision and 2 bytes for each list ($3 \times 5 = 15\text{bits} \simeq 2\text{bytes}$), as depicted in Figure 2. In this example, we see that the first byte is used to give the precision (3 bits) and then the remaining 5 integers are stored each in 3-bit precision taking a total of 15 bits. The result is fitted into bytes so a final bit is not used. The second list is processed in the same way but consists of values that are relative to the start index, which is in effect the length of the tokens.

3.4 Indexed data

Teanga supports the use of strings as a datatype, this can include annotations like lemmatizations or feature tags, which are often very repetitive. In order to avoid duplication, Cuaċ supports an indexing strategy that stores data in an index based on the order of occurrence. At the first occurrence of any string it is always stored as a string compressed by a method as described in Section 3.2, the string is then added to a Least-Recently Used (LRU) cache, with a hard-coded size of 1,000,000. If the string is seen again and is already in the LRU cache then it is assigned an index assigned from zero incrementally, all future occurrences will now be serialized using this index. In order to distinguish between strings and integer indexes when deserializing a second list is created with a single bit per index that indicates whether the next value is a string or not. The string values give the size of the string before the start of the string (instead of a null-terminator). The size of string and the indexes are stored as *variable-width* integers: these use the first bit of the integer to indicate whether further bytes are used for the representation of the number, and as such 1 byte is used to represent numbers up to 127 ($= 2^7-$), 2 bytes for numbers up to 16,383 ($= 2^14 - 1$) up to 5 bytes to be used for numbers

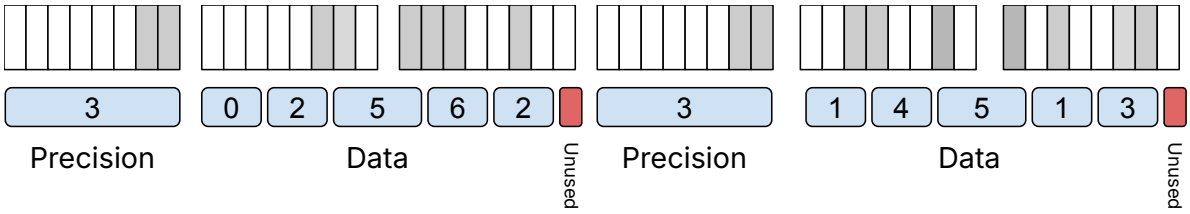

Figure 2: An example of the storage of indexes according to the Cuać data model

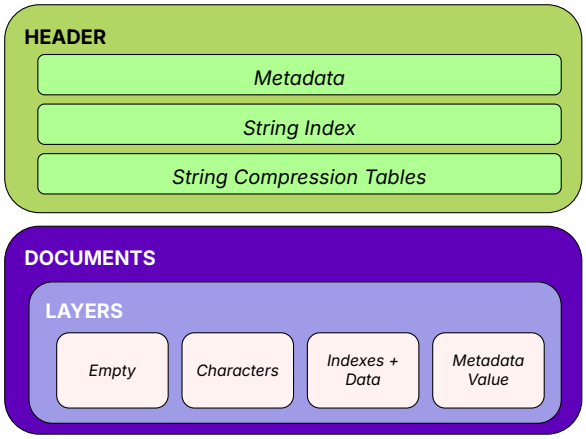

Figure 3: The structure of a Cuać file. A header gives metadata about the corpus as well as the string index for indexed data and the tables for string compression. Then each document is given layer by layer, where each layer is either absent, a compressed string, indexes and data (see Section 3.3 and 3.4) or a metadata as a key-value pair

in 32-bit normal precision. In Figure 4, an example of the representation of a string of values is given. First, the total length of the list is given as a variable-width integer, in this case, the number 200 is in the range of 2 bytes so two numbers (129, 72) are required to store it. Then, the indicator bits are stored that indicate whether the next data element is encoded as a string or is an index in the cache. Then the data is given, either by compressed string preceded by the length of the string in variable-precision[3] or a variable-width number giving the position of the word in the index.

A summary of the structure of a Cuać file is given in Figure 3.

4 Results

To evaluate the effectiveness of the Cuać format we evaluate it on two main measures. Firstly, we consider the file sizes and show that these are reduced by the use of this model. Secondly, we consider the time to convert a file into and from Cuać. This is important as it shows that the format does not introduce significant runtime overheads compared to using a more verbose format, and in fact, shows that our more compressed model is faster to read than the uncompressed version.

As a baseline, we consider the formats proposed for Teanga in McCrae et al. (2024) serialized in either YAML or JSON. As these are formats meant for human consumption, we also compare to Parquet (Kestelyn, 2013) as a binary format that is used by many projects including Hadoop, and Pandas. We use the Python implementation of Arrow[4] to convert data into this format.

In addition, we consider the effect of further compressing files using lossless compression algorithms. We consider the following algorithms:

DEFLATE DEFLATE (Deutsch, 1996a) is a lossless compression algorithm that combines LZ77 and Huffman coding, used in formats like gzip and PNG to efficiently reduce file sizes.

ZSTD Zstandard (Collet, 2018) is a fast, lossless compression algorithm, introduced by Meta, that provides high compression ratios and low latency, making it efficient for real-time and large-scale data compression.

BW The Burrows-Wheeler Transform (Burrows and Wheeler, 1994, BWT) is a reversible block-sorting compression algorithm that improves redundancy for better entropy encoding, forming the core of bzip2, which enhances compression efficiency with BWT, Huffman coding, and run-length encoding.

We evaluate our methodology across a wide range of corpora. Firstly, we evaluate a small section of the Colossal Common Crawl Corpus (Raffel

[3]The string length is given in preference to the null termination, so that the null character is available for text compression

[4]https://arrow.apache.org/docs/python/index.html

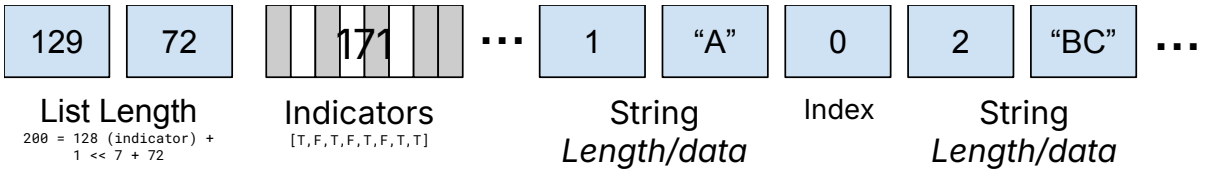

Figure 4: An example of the storage of a data string in Cuać. The data is a list of 200 string values starting with `["A", "A", "BC", ...]`

et al., 2019, C4) in a section identified as English web texts. We chose the first file from this corpus for our experiments and applied linguistic analysis with SpaCy[5] to this corpus. Secondly, we have converted all the NLTK corpora (Bird et al., 2009) into the Teanga format[6]. This collection includes a wide range of corpora from plain-text corpora to tagged and parsed corpora. These are primarily English-language corpora but also include corpora in a wide range of languages. We also converted the Universal Dependencies (Zeman et al., 2024) corpora collection in its entirety, which is a homogeneous corpus collection in terms of its annotations, however, it covers a wide range of languages and is useful to study the performance of the system across languages. Lastly, we converted the XL-WA (Martelli11 et al., 2023) and the Europarl Corpus (Koehn, 2005) to the Teanga format. Both datasets are provided in parallel text formats. XL-WA is a manually-curated benchmark for word alignment, covering 14 language pairs, including English and languages such as Arabic, Chinese, and Russian. It includes gold-standard word alignment annotations. For the Europarl Corpus, which is derived from European Parliament proceedings and offers parallel texts across 21 European languages, we generated word alignment annotations using `fast_align` (Dyer et al., 2013).

In Table 1, we present the size of the various corpora in four different serializations and with one of the three lossless compression algorithms. We present the original size of the corpora as distributed by NLTK and UD in uncompressed format, although we note that this includes documentary content that would not be captured in the Teanga data model. Further, the file sizes of the NLTK corpora are overall smaller as distributed than Teanga, due to the lack of explicit tokenization information, which is required in Teanga. For the uncompressed YAML version of the file, we present the total size in Table 1, all other sizes are specified relative to this file as a percentage. We see that across all corpora the Cuać format is smaller than the YAML and JSON and much smaller than the Parquet format. Further, we see that these reductions are further improved by the use of lossless compression, in some cases[7] the BW (`bzip2`) compression on the YAML or JSON actually achieves smaller file sizes than using Cuać, this is due to the text compression being used and it is important to note that BW compression is substantially computationally more expensive than the other methods considered here.

In order to measure the effect of text compression presented in Section 3.2, we compare the size of compressed files on the UD corpora. Again we present the absolute file size for the serialization without compression and the relative size for other compression methods. We see that for the English corpus, all methods provide similar size reduction in file sizes. For other languages using the Latin script, we generally see that the default tables provide effective text compression for most languages, except for some languages such as Turkish and Vietnamese that are typologically different from English. We also present Old Irish as an example of a very small corpus, and see that for this small corpus, generating a language-specific table introduces more overhead than reduction in file size. For non-Latin languages, we see that the default tables are not suitable and can substantially increase the file size. We see that it is effective to generate a language-specific table for languages that use a small set of letters, such as Arabic or Russian. However, for languages with a large number of characters, such as Chinese and Japanese, the short text compression is not able to reduce the length of the strings. Finally, we again see that no approach is effective for a small corpus language, namely Bengali, that does not use the Latin alphabet.

Finally, we consider the time to convert the resources into and from the Cuać format. In Table 3,

[5] https://spacy.io/

[6] https://teanga.io/corpora/

[7] Europarl and XL-WA in Table 1

Corpus	Original	YAML				JSON			
		None	DEFLATE	ZSTD	BW	None	DEFLATE	ZSTD	BW
C4	n/a	181.9 MB	24.18%	23.06%	15.59%	73.16%	22.50%	19.88%	14.65%
NLTK	359.0 MB (70.7%)	507.8 MB	25.39%	22.96%	16.82%	88.42%	22.40%	19.94%	15.24%
UD	2.72 GB (100.3%)	2.72 GB	16.16%	18.17%	8.49%	91.41%	14.19%	15.28%	7.74%
Europarl	1.5GB (64.1%)	2.34GB	25.82%	25.37%	13.10%	89.76%	24.34%	23.49%	12.61%
XL-WA	n/a	10.3 MB	21.46%	22.29%	11.63%	92.84%	20.09%	20.85%	11.25%
		Cuać				Parquet			
		None	DEFLATE	ZSTD	BW	None	DEFLATE	ZSTD	BW
C4	-	15.48%	12.13%	12.50%	11.16%	79.14%	31.14%	28.04%	25.61%
NLTK	-	36.16%	14.14%	13.44%	11.16%	253.57%	52.47%	45.84%	38.39%
UD	-	20.12%	9.36%	9.26%	7.42%	264.58%	57.47%	54.14%	40.18%
Europarl	-	26.67%	18.19%	17.71%	14.57%	418.07%	85.78%	76.53%	56.18%
XL-WA	-	28.60%	16.81%	15.21%	13.16%	540.96%	90.84%	86.37%	60.91%

Table 1: File Sizes of Test Corpora in Megabytes when Serialized with Formats. The sizes are presented relative to the YAML version with no compression.

Corpus	None	Smaz	Shoco (Default)	Shoco (Generate)
English	10.1 MB	84.96%	87.97%	84.88%
Catalan	3.9 MB	79.98%	86.48%	72.45%
Czech	34.5 MB	101.32%	102.07%	86.69%
French	4.0 MB	85.70%	89.26%	77.43%
German	50.6 MB	87.02%	88.66%	82.02%
Icelandic	20.2 MB	99.21%	101.57%	85.94%
Italian	6.3 MB	76.09%	81.73%	71.71%
Latin	18.2 MB	89.78%	88.90%	83.78%
Norwegian	9.5 MB	90.10%	92.31%	84.38%
Portuguese	11.2 MB	82.55%	87.98%	74.49%
Spanish	9.0 MB	81.61%	87.07%	74.74%
Turkish	12.9 MB	100.11%	99.57%	84.24%
Vietnamese	1.0 MB	107.12%	117.80%	91.61%
Old Irish	27.3 KB	100.63%	90.72%	126.24%
Arabic	4.1 MB	119.35%	158.54%	71.97%
Bulgarian	3.5 MB	109.28%	149.13%	80.05%
Chinese	6.1 MB	107.83%	117.39%	117.68%
Hebrew	2.4 MB	117.53%	180.62%	73.88%
Hindi	20.2 MB	109.79%	123.05%	94.63%
Japanese	49.3 MB	107.80%	130.74%	107.54%
Korean	22.3 MB	103.46%	118.98%	103.14%
Persian	11.7 MB	111.46%	145.80%	79.92%
Russian	33.9 MB	108.61%	153.17%	78.86%
Bengali	18.9 KB	107.29%	118.91%	124.92%
Average	-	98.69%	112.43%	88.05%

Table 2: Comparison of Text Compression Algorithms by File Size

Format	C4	Brown	Twitter	UD
JSON → YAML	8.8s	2.1s	9.5s	3m48.3s
JSON → Cuać	6.6s	1.6s	6.7s	3m49.2s
JSON → JSON	4.6s	1.0s	6.5s	3m39.6s
YAML → JSON	13.5s	3.1s	5.7s	4m16.8s
Cuać → JSON	4.1s	0.4s	3.4s	4m07.8s

Table 3: Time To Convert a Document to and from JSON

we measure the conversion of 5 corpora into and from JSON, where JSON is used as a consistent variable. As a baseline an idempotent translation of JSON to JSON is used, i.e., the JSON is fully deserialized and then fully serialized. We then consider the translation to and from Cuać and JSON. The results show that the conversion from Cuać is in most cases similar to that of JSON and faster than conversion to YAML, even with the extra complexity of the format and, in fact, in some cases we see that the conversion from Cuać is even faster than the JSON format. We primarily attribute this due to less IO operations due to the shorter files.

5 Discussion

The Cuać tooling is developed in Rust and in addition is compiled to Web Assembly (Rossberg, 2025), so it can be run on any platform or language using a tool like Wasmer[8]. This allows corpora to be used with the Teanga library, giving a data science interface for corpora similar to how Pandas uses Parquet as the underlying data storage. Further, it is used as the core of a database engine for Teanga corpora which is currently under development. The Cuać format is ideal for such a format as it allows data to be accessed quickly without overly burdening the database engine with a large amount of data. In particular, the use of text compression technologies is designed to still allow full-text search over the corpus due to the nature of the compression, in a way that would not be possible with the other lossless compression algorithms.

[8] https://wasmer.io/

As corpora in NLP grow larger and larger more efficient methods for handling such corpora are required. Large-scale NLP datasets (e.g., Common Crawl, Wikipedia, or domain-specific corpora) take up terabytes of space. A specialized compression format can significantly reduce storage requirements, making dataset management more cost-effective. Loading large text datasets from disk or transferring them over networks can be a bottleneck. A compressed format optimized for fast decompression can accelerate data loading, benefiting both training and inference workflows.

6 Conclusion

In this work, we have presented Cuać, a format for efficient and compact representation of large-scale corpora. Cuać substantially reduces the storage requirements while maintaining fast processing speeds. Our evaluation demonstrated that Cuać achieves superior compression better or similar to applying lossless compression to human-readable formats and strongly outperforming other binary formats not designed for corpus information. In this way, Cuać will enhance the computational efficiency of NLP applications and improve processing speeds across a range of NLP and machine learning applications. Future work will explore further optimization to target a wider range of corpora including multimodal corpora, when support for multimodal corpora is added to the Teanga data model.

Limitations

This work presents an analysis of the Cuać format across a wide range of corpora and languages, however this is not a complete evaluation across all possible corpora, so these results may not work in certain situations. We also note that the Teanga data model only supports plain text and annotated corpora and this method is not applicable to multimodal corpora. Finally, the computation times results show some variance and so depending on the encoding of the corpus, in some situations there may be increases in computational time associated with the use of the Cuać format.

Ethics Statement

There are no ethical issues with this work.

Acknowledgments

This research is supported by Taighde Éireann - Research Ireland under Grant Number SFI/12/RC/2289_P2 Insight_2, Insight SFI Centre for Data Analytics and Grant Number 13/RC/2106_P2, ADAPT SFI Research Centre.

References

Jyrki Alakuijala, Evgenii Kliuchnikov, Zoltan Szabadka, and Lode Vandevenne. 2015. Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM and Bzip2 compression algorithms. Technical report, Google Inc.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*, 1st edition. O'Reilly Media, Inc.

Nieves R Brisaboa, Ana Cerdeira-Pena, Guillermo de Bernardo, Antonio Fariña, and Gonzalo Navarro. 2023. Space/time-efficient rdf stores based on circular suffix sorting. *The Journal of Supercomputing*, 79(5):5643–5683.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City. Association for Computational Linguistics.

Michael Burrows and David J. Wheeler. 1994. A block sorting lossless data compression algorithm. Technical Report Technical Report 124, Digital Equipment Corporation. Archived from the original on January 5, 2003.

Yann Collet. 2018. Zstandard compression and the application/zstd media type. Technical Report RFC 8478, Internet Engineering Task Force. Retrieved 7 October 2020.

Lasse Collin. 2010. Xz utils. `https://tukaani.org/xz/`. Accessed: 2025-03-02.

P. Deutsch. 1996a. Deflate compressed data format specification version 1.3. Technical report, RFC 1951.

P. Deutsch. 1996b. Gzip file format specification version 4.3. Technical report, RFC 1952.

Chris Dyer, Victor Chahuneau, and Noah A Smith. 2013. A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 644–648.

Javier D Fernández, Miguel A Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. 2013. Binary RDF representation for publication and exchange (HDT). *Journal of Web Semantics*, 19:22–41.

Antonio Hernandez-Illera, Miguel A. Martinez-Prieto, and Javier D. Fernandez. 2015. Serializing RDF in compressed space. In *2015 Data Compression Conference*, page 363–372. IEEE.

Antonio Hernández-Illera, Miguel A. Martínez-Prieto, and Javier D. Fernández. 2020. RDF-TR: Exploiting structural redundancies to boost RDF compression. *Information Sciences*, 508:234–259.

Nancy M Ide and C Michael Sperberg-McQueen. 1995. The TEI: History, goals, and future. *Computers and the Humanities*, 29:5–15.

Justin Kestelyn. 2013. Introducing Parquet: Efficient columnar storage for Apache Hadoop - Cloudera engineering blog. Archived from the original on 2013-05-04. Retrieved 2018-10-22.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of machine translation summit x: papers*, pages 79–86.

Federico Martelli11, Andrei Stefan Bejgu, Cesare Campagnano11, Jaka Čibej14, Rute Costa10, Apolonija Gantar14, Jelena Kallas, Svetla Koeva, Kristina Koppel, Simon Krek, et al. 2023. Xl-wa: a gold evaluation benchmark for word alignment in 14 language pairs.

John P. McCrae, Priya Rani, Adrian Doyle, and Bernardo Stearns. 2024. Teanga data model for linked corpora. In *Proceedings of the 9th Workshop on Linked Data in Linguistics @ LREC-COLING 2024*, pages 66–74, Torino, Italia. ELRA and ICCL.

Raffaele Perego, Giulio Ermanno Pibiri, and Rossano Venturini. 2021. Compressed Indexes for Fast Search of Semantic Data . *IEEE Transactions on Knowledge & Data Engineering*, 33(09):3187–3198.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*.

Andreas Rossberg. 2025. WebAssembly specification. W3c community group report, W3C.

Salvatore Sanfilippo. 2009. Smaz - compression for very small strings. `https://github.com/antirez/smaz`. Accessed: 2025-03-02.

Christian Schramm. 2014. Shoco: A fast compressor for short strings. `https://github.com/Ed-von-Schleck/shoco`. Accessed: 2025-03-02.

Tangina Sultana, Md. Delowar Hossain, Md Golam Morshed, Tariq Habib Afridi, and Young-Koo Lee. 2024. Inductive autoencoder for efficiently compressing RDF graphs. *Information Sciences*, 662:120210.

Tangina Sultana and Young-Koo Lee. 2022. gRDF: an efficient compressor with reduced structural regularities that utilizes gRePair. *Sensors*, 22(7):2545.

Ruoyu Wang, Raymond Wong, and Daniel Sun. 2024. Estimation-based optimizations for the semantic compression of RDF knowledge bases. *Information Processing & Management*, 61(5):103799.

Mark D. Wilkinson et al. 2016. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 3(1).

Daniel Zeman et al. 2024. Universal dependencies 2.15. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.