EFFICIENT LEXICAL ENCODING OF NATURAL LANGUAGE

Brian L. Heers

InTech Collegiate Academy

Ms. Davidson

ISEF 2025

Contents

Engineering Plan

Rationale

Incredible advances in the field of data storage and transmission have undoubtedly changed the way data is seen. From 56kb/s dial-up 20 years ago to 8gb/s fiber today, even home internet has reached speeds that makes one wonder how to use all that bandwidth. Most people no longer consider the size of a file before downloading it off the web. Data transmission has transitioned from efficient, optimized, data-dense encoding to simpler, lazier and more wasteful methods, as the bandwidths have increased exponentially.

However, there are still countless modern applications that have not or are unable to keep up with bandwidth standards, and still require efficient and optimized methods. Long-distance radio transmissions, especially satellite communications and other fields of space exploration, lack the luxury of multiple gigabits of cheap bandwidth. In mediums with elevated interference, and therefore slower effective transmission speeds, such as: metal wire in EMI environments; radio in foggy or rainy weather, through walls, or noisy RFI environments; or more unique scenarios like sonar-based underwater wireless communications, alternative data transmission protocols are necessary. Even cell communication, a relatively high-speed application, operates on such a massive scale that large benefits can arise from more efficient data transmission. Many of these data transmission applications use natural language (NL) text as their media and could benefit from an encoder more efficient than ASCII, the current method.

Engineering Goal

To propose a solution to constrained bandwidths like those mentioned above, this project entails the development of a program that encodes and decodes text to and from binary, optimized for NL. It will function by assigning identifiers to lexemes (a unit of language: one word, two words, phrase, punctuation, abbreviation, letter), arranged to maximize the information density and minimize the bit length of encoded data.

Engineering Outcome

The goal is to optimize the encoding of NL. Optimizations will be continuously improved upon with successive iterations of the NL encoder. Its performance can be easily measured by calculating how many times more efficient it is than ASCII.

Additionally to metrics used to determine if the goal is met, the project can be deemed unsuccessful if it is:

- Less efficient than ASCII (longer binary encodings)
- Provides unusable results, or is malfunctional
- Too lossy and semantics are changed or missing

The different goals for this project are to have the NL encoder be either:

- More efficient than ASCII
- 1.5 times as efficient
- 2 times as efficient
- 3 times as efficient

Materials

- Computer
- VS Code
- Internet access
- p5.js Library (in case of graphical representation)

Procedure

1. Obtain materials
2. Find and download a word frequency dataset
3. Write a program that assigns decimal indexes from the dataset to words in a text
4. Create alternative measures for words that are not in the dataset. Possible measures include:
    a. Split into letters
    b. Split into morphemes
        i. Prefixes, roots, and suffixes
    c. Autocorrect to correctly spelled words contained in the dataset
    d. Change infrequent words to more frequent synonyms
5. Create an efficient method of turning the array of indexes into one-dimensional binary that favors more frequent words
    a. The method cannot be fixed-length encoding that treats all words equally
6. Support punctuation
    a. Order punctuation by frequency in a specific corpus
    b. Limit options for punctuation for minimal bit-length
7. Support more complex lexemes, such as
    a. Phrases
    b. Idioms
    c. Sets of words that often go together
8. Evaluate the pros and cons compared to ASCII
9. Optimize the current system by adding new functionality aimed at compressing data

10. Evaluate the new iteration compared to the previous iteration
11. Return to step 8

Risk and Safety

Being a project fully taking place on a computer, only a few risks could be associated with it. Fatigue and eye strain could have the potential to become a problem and could be mitigated by breaks, if necessary.

Bibliography

Chen, T., & Kan, M.-Y. (2013). Creating a Live, Public Short Message Service Corpus: The NUS SMS Corpus. *Language Resources and Evaluation*, *47*(2)(2013), 299–355. https://doi.org/10.1007/s10579-012-9197-9

Khalid Sayood. (2000). *Introduction to data compression* (2nd ed.). Morgan Kaufmann Publishers.

Mahmood, Md. A., & Hasan, K. M. A. (2019, November 1). *A Dictionary based Compression Scheme for Natural Language Text with Reduced Bit Encoding*. IEEE Xplore; IEEE. https://doi.org/10.1109/RAAICON48939.2019.62

Mahmood, Md. A., & Hasan, K. M. A. (2022). An Efficient Compression Scheme for Natural Language Text by Hashing. *SN Computer Science*, *3*(4). https://doi.org/10.1007/s42979-022-01210-0

Piantadosi, S. T. (2014). Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic Bulletin & Review*, *21*(5). https://doi.org/10.3758/s13423-014-0585-6

Tatman, R. (2017). *English Word Frequency*. Kaggle. https://www.kaggle.com/datasets/rtatman/english-word-frequency/data

Engineering Paper

Rationale

Incredible advances in the field of data storage and transmission have undoubtedly changed the way data is seen. From 56kb/s dial-up 20 years ago to 8gb/s fiber today, even home internet has reached speeds that make one wonder how to use all that bandwidth. Most people no longer consider the size of a file before downloading it off the web. Data transmission has transitioned from efficient, optimized, data-dense encoding to simpler, lazier and more wasteful methods, as the bandwidths have increased exponentially.

However, there are still countless modern applications that have not or are unable to keep up with bandwidth standards and still require efficient and optimized methods. Long-distance radio transmissions, especially satellite communications and other fields of space exploration, lack the luxury of multiple gigabits of cheap bandwidth. In mediums with elevated interference, and therefore slower effective transmission speeds, such as: metal wire in EMI environments; radio in foggy or rainy weather, through walls, or noisy RFI environments; or more unique scenarios like sonar-based underwater wireless communications, alternative data transmission protocols are necessary. Even cell communication, a relatively high-speed application, operates on such a massive scale that large benefits can arise from more efficient data transmission. Many of these data transmission applications use natural language (NL) text as their media and could benefit from an encoder more efficient than ASCII, the current standard for efficiently encoding NL.

Engineering Goal

To propose a solution to constrained bandwidths like those mentioned above, this project entails the development of a program that encodes and decodes text to and from binary, optimized for NL. It will function by assigning identifiers to lexemes (a unit of language: one word, two words, phrase, punctuation, abbreviation, letter), arranged to maximize the information density and minimize the bit length of encoded data.

Procedure

Same as in the Engineering Plan (Page 8).

Development and Iteration Write-Ups

(Following pages)

First Iteration Write-Up

Foundation

The foundation of this project is assigning IDs to words and having more frequent words take up less bits when encoded. Consider the first 1000 most frequent words in English Wikipedia[1] arranged as an array in JavaScript (Figure 1). Words that aren't an element of the dataset must also be accounted for and are contained in an array of specialIDs made up of the basic Latin alphabet (a-z), the numbers (0-9), and punctuation (. , - ' " $ ( ) : ? / ! & ; \n).

Figure 1 – Sample of wordfreq.js

```
const words = [
    "the",
    "of",
    "and",
    ...
    "spread",
    "suggested",
    "Paul",
]
```

Encoding Process (Proof of Concept)

Use Figure 2 as a guide to understanding the following paragraphs.

Figure 2 – Each step in the encoding process

```
Original    You              must        go               to          school
Decimal ID  311              214         662              50          422
Binary      0001 0011 0111 - 1101 0110 - 0010 1001 0110 - 0011 0010 - 0001 1010 0110
Nybbles     3 (011)          2 (010)     3 (011)          2 (010)     3 (011)

Index                End   IDs
011 010 011 010 011 000 - 0001 0011 0111 | 1101 0110 | 0010 1001 0110 | 0011 0010 | 0001 1010 0110
```

The sentence, `"You must go to school"`, contains only elements of the dataset. The first step in Figure 2 is giving each word an ID. The ID of a word is its index in the words array plus the

[1]https://en.wiktionary.org/wiki/Wiktionary:Frequency_lists/English/Wikipedia_(2016) Acquired 9/20/24

length of specialIDs. This results in the first 52 IDs being assigned to specialIDs and the rest to words. Next, they are converted to binary (ignoring the leading zeroes for now). Each word is now represented in binary, and the sentence could be represented as:

100110111110101101010010110110010110100110

(1 0011 0111 | 1101 0110 | 10 1001 0110 | 11 0010 | 1 1010 0110)

There is however one problem with this. When trying to decode this string, a decoder would not know where each word starts and ends because of the variable bit-length of each word.

To fix this, an index is added to the beginning of the datastream, which specifies the bit-length of each word. This is where the leading zeroes come in: Each binary representation of the ID will have enough leading zeroes to make it divisible by four i.e. be able to count in whole numbers the number of nybbles (byte but for four bits) long it is. Given that an ID can't have a nybble-length of 0, a stop code of 000 will be used to inform the decoder that the index has ended, and the IDs have begun. This can be seen in the bottom row of Figure 2, where the Nybbles row and the Binary row combine. Now, the following uniquely decodable binary code can be used to represent the sentence `"You must go to school"`:

0110100110100110000001001101111101011000101001011000110010000110100110

Compared to the ASCII encoding:

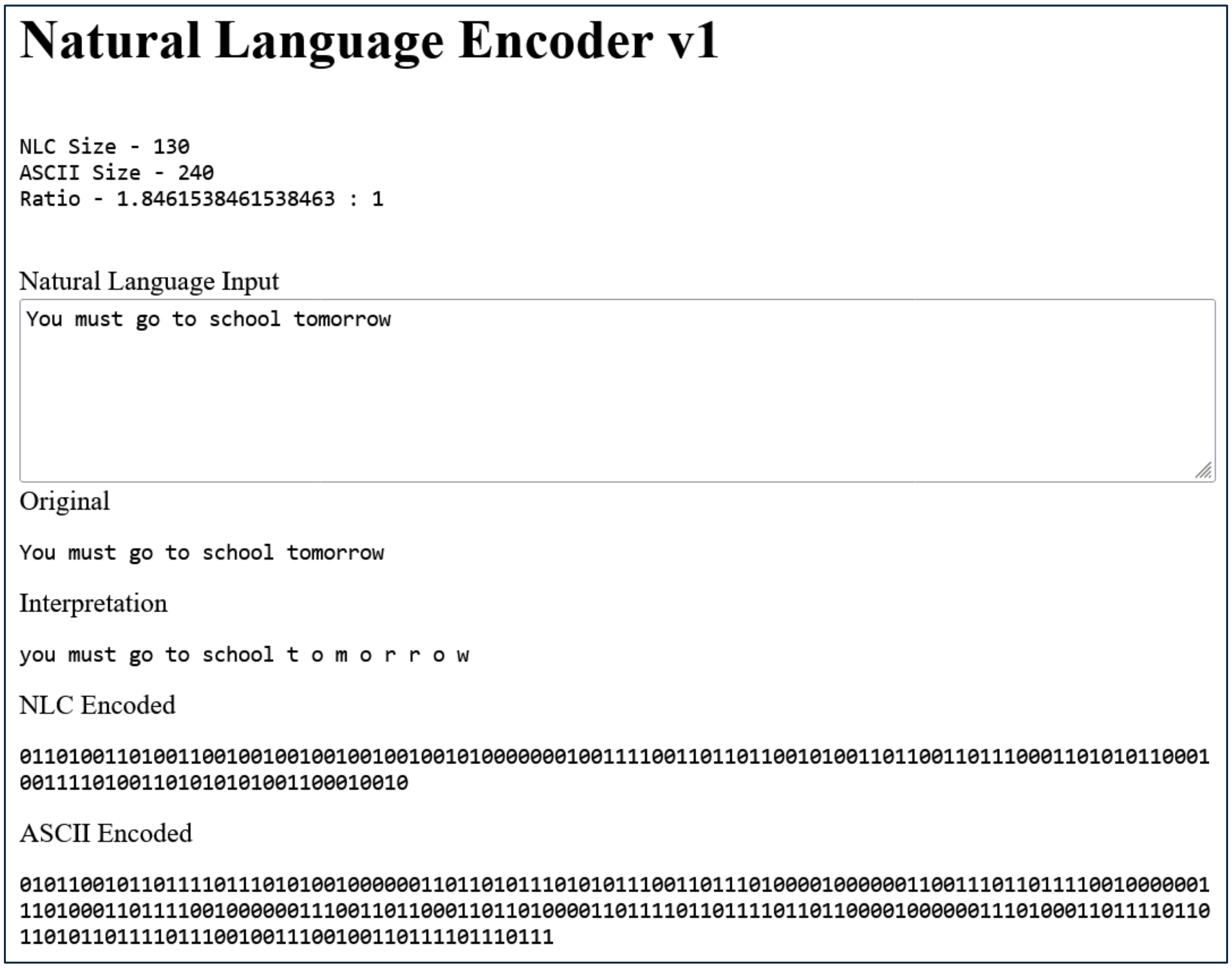
01011001011011110111010100100000011011010111010101110011011101000010000000110011
10110111100100000001110100011011110010000000111001101100011011010000110111101101 1
1101101100

For this sample, the NLC (Natural Language Compression) encoding is 2.4 times more efficient than ASCII's. The next step is automating this process in a JavaScript program, so that more and larger tests can be conducted.

Implementation

The program's complete source code can be found on page 66. The UI of the web interface is shown in Figure 3.

Figure 3 – UI of the web interface



Natural Language Encoder v1

```
NLC Size - 130
ASCII Size - 240
Ratio - 1.8461538461538463 : 1
```

Natural Language Input

```
You must go to school tomorrow
```

Original

```
You must go to school tomorrow
```

Interpretation

```
you must go to school t o m o r r o w
```

NLC Encoded

```
0110100110100110010010010010010010010100000001001111001101101100101001101100110111000110101011000 1
0011110100110101010100110001 0010
```

ASCII Encoded

```
01011001011011110111010100100000011011010111010101110011011101000010000001100111011011110010000001
11010001101111001000000111001101100011011010000110111101101111011011000010000001110100011011110110
1101011011110111001001110010011011110111 0111
```

The main script is run every time a key is pressed in the text-area (this will be called a refresh), so that the user gets real time feedback on how their text is being encoded. Figure 4 visualizes the script's behavior and Figure 5 shows the process intuitively on an example sentence.

Figure 4 – Automated process, visualized

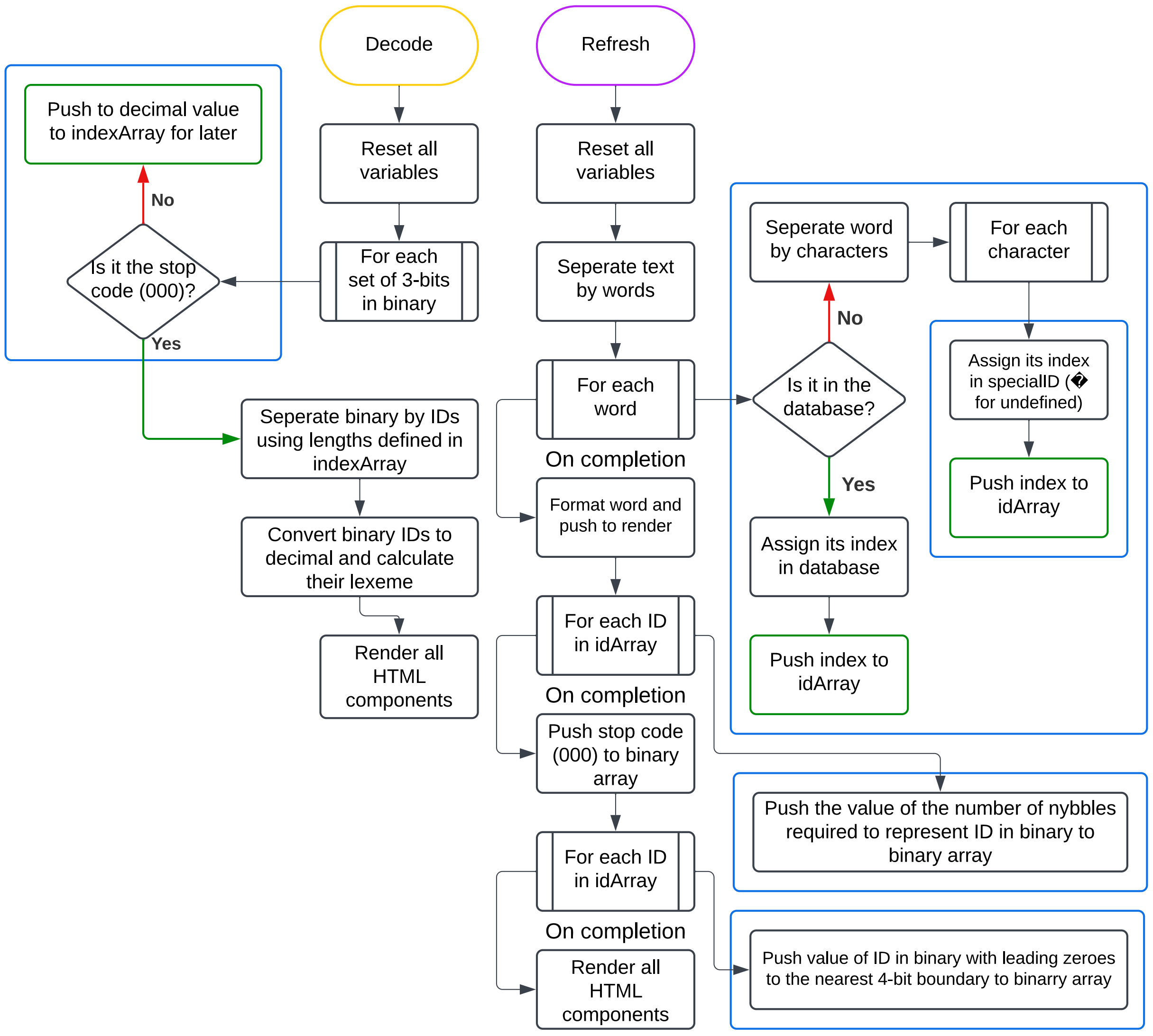
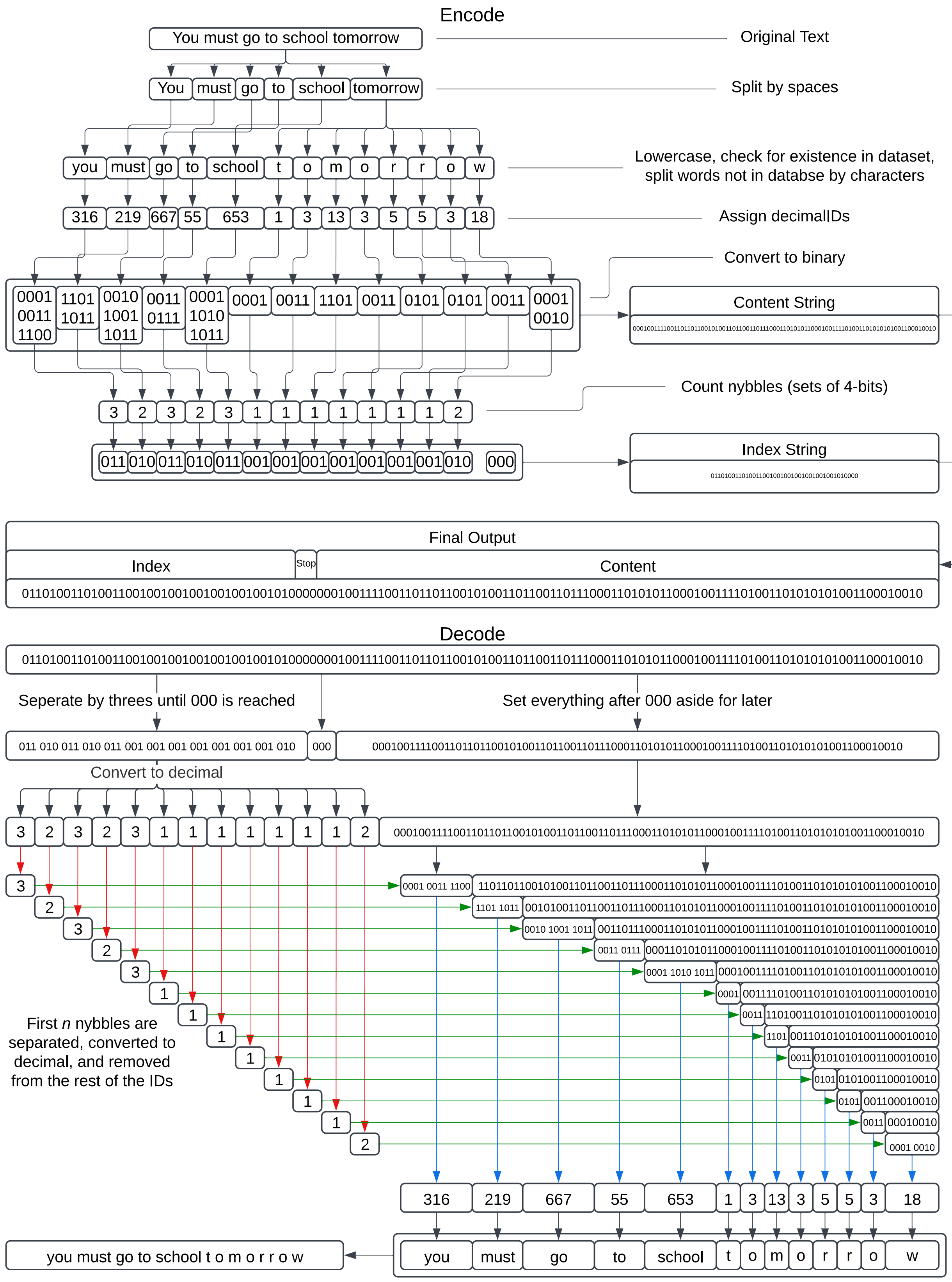

Decode
Refresh
Push to decimal value to indexArray for later
Reset all variables
Reset all variables
No
Is it the stop code (000)?
For each set of 3-bits in binary
Seperate text by words
Seperate word by characters
For each character
Yes
No
For each word
Is it in the database?
Assign its index in specialID (� for undefined)
Seperate binary by IDs using lengths defined in indexArray
On completion
Format word and push to render
Yes
Push index to idArray
Convert binary IDs to decimal and calculate their lexeme
Assign its index in database
Render all HTML components
For each ID in idArray
Push index to idArray
On completion
Push stop code (000) to binary array
Push the value of the number of nybbles required to represent ID in binary to binary array
For each ID in idArray
On completion
Push value of ID in binary with leading zeroes to the nearest 4-bit boundary to binarry array
Render all HTML components

Figure 5 – Encoding and decoding sequence on example sentence



In the first step, the original text is split by spaces into words. Because of the little information the capitalization of a letter in a word conveys, all the words are lowercased before being checked for existence in the dataset. If a word *is* in the dataset, it is replaced with its ID, which is a unique number that can represent letters, numbers, punctuation, and words. If a word is *not* in the dataset, or is touching punctuation[2], it is split into individual letters, which are then assigned IDs. Next, the IDs are converted from decimal into binary with enough leading zeroes to make the bit-length divisible by four. This results in the binary values being whole nybbles (4-bits) long. This is required for the next step, where the bit-length is divided by four to return the nybble-count of each binary ID. That nybble count is then converted to binary at a fixed length of 3-bits per index assignment. The final encoded output is the concatenation of the index binary, the stop code (000), and of the ID binary.

Decoding is a simpler process. The algorithm pushes the first three bits of the input to an index array and continues pushing the consecutive sets of three bits until it reaches the stop code (000). The index array is converted back into decimal for simpler processing. For each index assignment in the index array, with *n* being the index assignment, the first *n* nybbles (sets of 4-bits) of the binary input after the stop code are converted to decimal, pushed to an ID array, and discarded from the rest of the binary. After the ID binary has been converted to an array of decimal IDs, they are looked up in the specialIDs array or in the words array and given their values. They are then joined, separated by spaces.

[2] Because a word is defined here as elements separated by spaces, `"Hello, world"` is split into `["Hello,","world"]`. `"Hello,"` wouldn't match with `"hello"`, because it is "touching" punctuation. It will therefore be split into `["h","e","l","l","o",","]`.

In the previous examples and explanations, the wiki-1k dataset was used for the words, but for better efficiency, a new dataset is implemented. web-333k[3] contains 333,333 words derived from the Google Web Trillion Word Corpus. This improves efficiency by using the overhead provided by the 3-bit index assignment lengths. A darkmode is also implemented by exporting the darkmode CSS from darkreader.org on firefox.

Data

4 text samples were encoded in NLC (binary encoded by the NL encoder) and to ASCII, and had their ratios calculated.

Table 1 – v1 Sample Text Testing

Great Gatsby	2.120
US Constitution	2.346
SMS	1.699
English News	2.100

Data Analysis

The first sample text is the popular public domain book, The Great Gatsby. The first chapter was encoded and compared, and NLC took up 2.12 times less space than ASCII. The second sample is the full US Constitution, which was 2.346 times as efficient. The higher efficiency can be attributed to the lower amount of punctuation than that in a book. The third sample contains messages from the 2015 NUS SMS Corpus[4]. This had the lowest efficiency out of the tests because

[3] https://www.kaggle.com/datasets/rtatman/english-word-frequency/data
[4] http://wing.comp.nus.edu.sg:8080/SMSCorpus http://link.springer.com/article/10.1007%2Fs10579-012-9197-9 or https://github.com/kite1988/nus-sms-corpus/raw/refs/heads/master/smsCorpus_en_xml_2015.03.09_all.zip

of the excessive use of punctuation and Romanized Chinese names and words. English News was acquired from Universität Leipzig's Wortschatz, under English News 2023[5].

Sources of Error

The program is consistent and all known bugs affecting output or statistical calculations have been addressed. There could remain undiscovered bugs or errors in calculations, but the results and tests remain consistent with the expected outcomes.

Variations

Many changes can be made to improve the efficiency and functionality of the encoder. To address the problem with the SMS test – an important problem as this project could be used for satellite-based personal communication – support for punctuation will be adjusted to eliminate its negative impact of splitting words that would normally be in the dataset. The color rendering for the input could be changed to reflect the actual efficiency of the words more accurately. Split words should be recombined (currently their letters are separated by spaces), and spaces and new lines should be represented more accurately (currently new lines are just a whitespace and any number of spaces together only represent at most one). A solution must address the overhead associated with structures of the program that were chosen arbitrarily: Binary is split by nybbles; index assignments are three bits; 333,333 entries in the dataset, even though 268,435,455 ($2^{4(2^3-1)}$) entries is the limit for three-bit nybble-length index assignments. Functionality should be added to allow the changing of lexeme datasets to optimize for different applications, such as between texting, official communications, informal communications, and the distinctive styles of communication between different age groups and demographics.

[5] https://wortschatz.uni-leipzig.de/en/download/English

Conclusion

The goal was to create an encoder for NL, more efficient than ASCII, and that goal was achieved. The encoder is consistently over 1.5x as efficient, even on the worst-case texts. Some semantic value is lost however; words not in the dataset are encoded as split by spaces for every character (`"helloo"` → `"h e l l o o"`), anything uppercased is encoded lowercased (`"Hello"` → `"hello"`), any amount of spaces more than one is only encoded as one (`"hello···world"` → `"hello·world"`), characters not in specialIDs are replaced with � (`"1 + 1 = 2"` → `"1 �1 � 2"`), and newlines are rendered as whitespace. There are also various other changes stated in the *Variations* section that will be addressed in the following iterations. A demo of the encoder is available on isef.heers.net/v1.

Second Iteration Write-Up

Changes

Iteration 2 (v2) is a minor augmentation of v1, meant to address most of its semantic-loss problems. Punctation and spacing handling has been completely redone, and any amount of punctuation can be placed together with any amount of spacing (`'It said, "Hello World!"'` is encoded maintaining `', "'` and `'!"'`). Newlines are now replaced with `"<br>"` when being passed to HTML, allowing for specification of newlines and arbitrarily long sequences of whitespaces are also supported.

A further resolved problem was words not in the dataset being split into individual characters. This was solved by specifying a space after a string of characters in an undefined word. All characters are then assumed to be grouped together until there is a specified space separating them.

There is still the problem of undefined characters being replaced with �, and the ampersand (&) character was removed from the supported punctuation to make room for an uppercase control character. The uppercase control character is treated as a punctuation mark and allows for the specification of uppercase letters.

For the webUI, the color scheming of the formatted text has been altered to show the efficiency of each given word. A dropdown menu was added to select between the 333k-web and 1k-wikipedia datasets, however, as of now there are no benefits to using the 1k-wikipedia dataset. A checkbox flag was also added to show “brackets”, allowing the user to better understand how the text is being encoded.

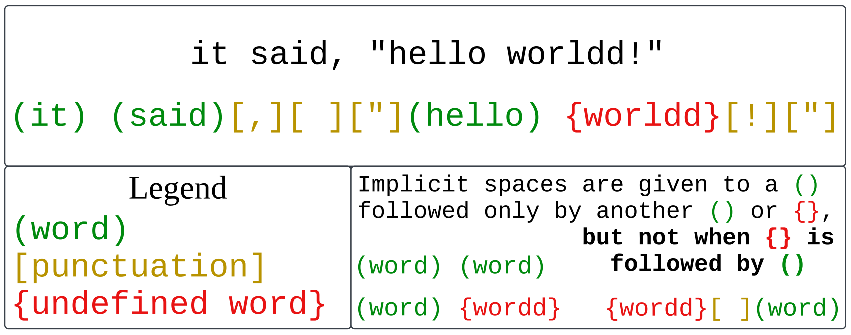
Punctuation and Spacing Handling

In v1, the input `'It said, "Hello World!"'` is interpreted as `'it s a i d , " h e l l o w o r l d ! "'`. In most cases, an English speaker could understand this, but for different sentences, or non-native English speakers, important context can be lost. A solution would have to allow arbitrary placement, count, and separation of punctuation, while keeping the spaces between words implied and not taking up additional data.

This is accomplished by parsing the original text first by its punctuation, instead of by its spaces. The encoder starts searching from the beginning of the text until a punctuation is reached; if it is a space, it is usually ignored (spaces between words), but if it is followed by another punctuation, it is encoded as an explicit space. If the reached punctuation is not a space, it is encoded as normal, and any number of punctuations can follow the first one and be encoded. This method of assuming spaces between words and specifying spaces around punctuation is referred to as *Implicit Spacing* and is shown in Figure 6.

When the text is later decoded, IDs that are words, which are followed by another word or undefined word, have an implicit space between them. All punctuation IDs are decoded without implied spaces but can still be separated explicitly: `'it said, "hello world!"'` would be interpreted exactly as `'it said, "hello world!"'`.

Figure 6 – Implicit Spacing Handling



it said, "hello worldd!"
(it) (said)[,][ ]["](hello) {worldd}[!]["]
Legend
(word)
[punctuation]
{undefined word}
Implicit spaces are given to a () followed only by another () or {},
but not when {} is followed by ()
(word) (word)
(word) {wordd}
{wordd}[ ](word)

Uppercase Handling

To allow for non-ambiguity in upper- and lowercased characters, an ability to specify an uppercase character is needed. ␑ is the ASCII Device Control One character (now referred to as XON). XON being a character, it is easy to implement into the code as a punctuation mark. To specify that a character is capitalized, XON is placed preceding the uppercase character and treated otherwise as punctuation mark: To use uppercasing, `"Hello"` is specified as `"␑hello"`. The currently supported punctuation is shown in the following Figure 7.

Figure 7 – v2 Supported Punctuation

```
[.] [,] [-] ['] ["] [$] [(] [)] [:] [?] [/] [!] [;] [\n] [ ] [␑]
```

To implement using XON as an uppercase control character, a regular expression is applied to the input string, which replaces every uppercase character with itself preceded by XON:

`/([A-Z])/g` replaced with `"␑$1"`

The whole input string is then converted to lowercase. Now, the encoder treats XONs as any other punctuation.

When an NLC binary string is decoded, everything is first decoded as normal, and then all XONs are removed while uppercasing the character after it.

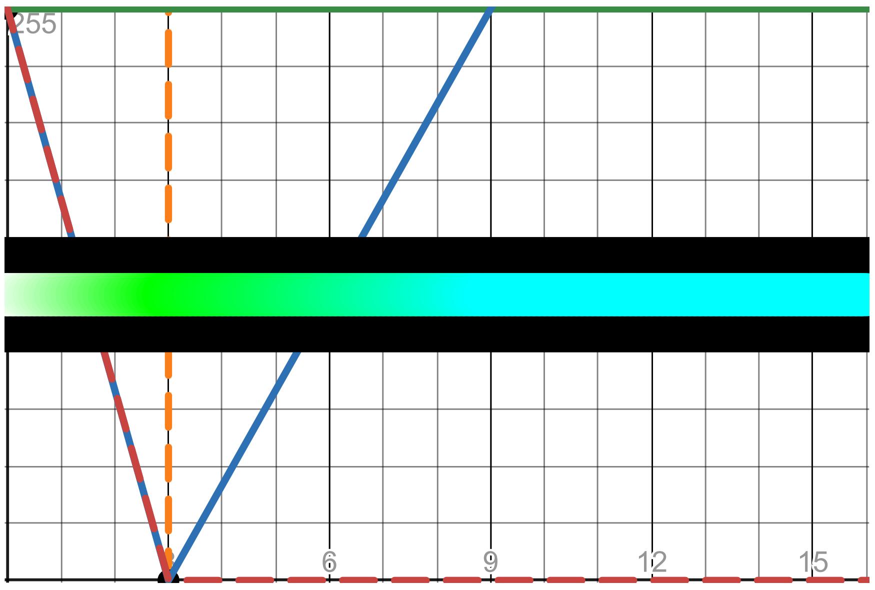
webUI Color Formatting

For this section, words are defined as elements of the dataset. Punctuation is any element of the array specified in the above Figure 7. Undefined words are any string of characters that are not an element of the dataset. The single letter words, including `"a"`, were removed from the dataset because of technicalities, and are therefore also undefined words. All elements containing numbers were also removed from the dataset, and cause words to be undefined when contained in them.

Punctuation is colored a yellow (#ffd700), and undefined words are colored pure red (#ff0000). Words have a specific coloring scheme to represent their effective encoded ratio.

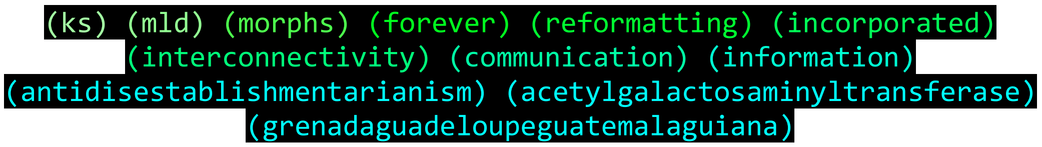
Each word in the dataset has a “ratio”. This is the ratio between how many bits it would take up by itself in NLC (without the index stop code) and in ASCII. For example, `"hello"` requires three content nybbles and three bits for its index assignment (all words have a three-bit index assignment). In total, it is 15 bits in NLC. In ASCII, its bit-length is the number of characters it is times eight, so `"hello"` is 40 bits in ASCII, and the ratio between the two is 2.67. The webUI displays each word in a color representative to its ratio. There are two variables used to control the look of the words. Low ratios are whiter and become greener as the ratio increases until the *blueThreshold*. After that, they become more cyan until *maxRatio*, where they stay at pure rgb(0, 255, 255). blueThreshold and maxRatio can be changed in the code and are purely aesthetic constants. Figure 8 shows the individual red, green, and blue channels’ values (y-axis) for different ratios (x-axis). It is using the actual blueThreshold and maxRatio used in this v2 source code, which are 3 and 9, respectively.

Figure 8 – Word Color Scheme Channel Graph



The ratios in the dataset range from 0.84 for some two-letter words to 13.47 for `"grenadaguadeloupeguatemalaguiana"`. The words with the highest ratios are primarily not words, but rather errors a web-scraper made while collecting the words. The most efficient word that is orthographically correct is `"acetylgalactosaminyltransferase"` at 10.78, followed by other chemical compounds. The most efficient non-chemical word is `"antidisestablishmentarianism"`. Examples of words of different ratios and how they appear on a dark background are shown in Figure 9. Notice that they do not become any more cyan after maxRatio.

Figure 9 – Words ranging from approximately 0.8, 1, 2, …, 8, 9.7, 10.8, 13.8 ratios respectively



```
(ks) (mld) (morphs) (forever) (reformatting) (incorporated)
(interconnectivity) (communication) (information)
(antidisestablishmentarianism) (acetylgalactosaminyltransferase)
(grenadaguadeloupeguatemalaguiana)
```

Decoding and Progressive-Computing

The webUI has two new pages: NLC Decoder and Progressive-Compute. The decoder is straightforward; an NLC binary string is entered, and natural language is outputted. Progressive-Compute, on the other hand, performs the same function as the encoder, but splits processing across different rendering frames. It is much slower and is meant to run on under-powered systems, but it is overall impractical and will be improved upon in v3.

Data

The web333k dataset, sorted by ratio, is available at isef.heers.net/v2/web333k-ratios.xlsx, however, the dataset contains words scraped from the web, and many words on the web are NSFW (Not Safe For Work). Only the first 25 NSFW words are censored, but if wished, their content be viewed in the formula bar.

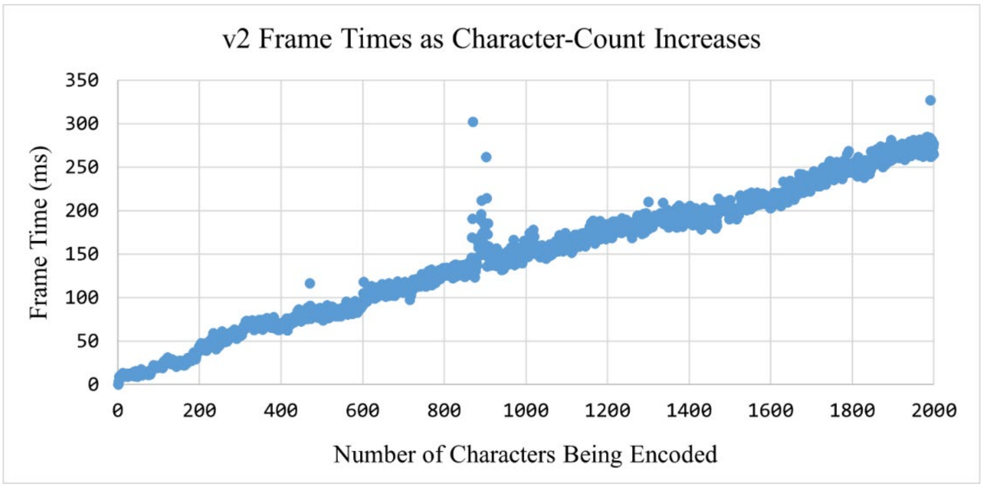
To test the efficiency of v2, 4 text samples were converted to NLC and to ASCII, and had their ratios calculated, as in the first iteration:

Table 2 – v2 Sample Text Testing

Text	Characters	NLC	ASCII	v2 Ratio	v1 Ratio	Link
US Constitution	26356	88480	210848	2,383	2,346	https://www.archives.gov/founding-docs/constitution-transcript
2023 English News	24756	91773	198048	2,158	2,1	https://wortschatz.uni-leipzig.de/en/download/English
The Great Gatsby (Chapter 1)	32890	103958	263120	2,369	2,12	https://www.gutenberg.org/cache/epub/64317/pg64317.txt
SMS	60000	269817	480000	1,779	1,699	https://github.com/kite1988/nus-sms-corpus
			Average	2,172	2,066	

Speed tests were performed on the model to calculate frame times for different numbers of characters in the 2023 English News sample. The data table is available also on isef.heers.net/v2/v2-testing.xlsx, and a scatter plot of its values is shown in Figure 10.

Figure 10 – Chart of v2 Frame Times as Character-Count Increases



Data Analysis

From the dataset, many of the top words are not actual words and will never be used in natural language. This causes problems by taking up space and skewing the indexing scheme to allow such high ratios that will never be achieved.

While there were improvements in the sample texts' compression ratios compared to v1, they were hindered by having to encode uppercasing and spacing, which v1 just ignored.

The exact frame times will vary between different systems executing the program, but it is seen that the frame time increases linearly with the number of characters being encoded. This is better than an exponential increase, which would cause problems with longer texts.

Sources of Error

As with the previous iteration, the program is consistent and all known bugs affecting output or statistical calculations have been addressed. There could remain undiscovered bugs or errors in calculations, but the results and tests remain consistent with the expected outcomes.

The sudden spike at approximately 880 characters in Figure 10 was caused by external factors influencing the available CPU time of the computer used during the test, but the irregularity does not affect the conclusion drawn from the data analysis.

Variations

The program still contains the overhead associated with arbitrarily chosen structures: binary is split by nybbles; index assignments are three bits; 333,333 entries in the dataset, even though 268,435,455 ($2^{4(2^3-1)}$) entries is the limit for three-bit nybble-length index assignments. There also is no functionality to allow the changing of lexeme datasets to optimize for different applications, such as between texting, formal and informal communications, and the distinctive styles of communication between different age groups and demographics.

In the next iteration, the entire indexing system will be redone to more effectively represent the dataset's composition. The index stop code (000) can be removed and the end of the index can be calculated based on the index assignments and the total length of the binary string.

Because 000 would no longer exist as an index assignment, it could be used to indicate some special function. For example, an index assignment of 000 could mean that the next word is accurately completable by a deterministic LLM, like auto suggestions on a phone keyboard, or an index assignment of 000 could indicate that the next *n* IDs are 5-bit letters, with *n* being defined by the number of 0s following the 000 and terminated by a 1 (000 0 0 1 could mean the next four IDs are letters). Similarly, an index assignment of 000 could be used to show that the next *n* IDs have the same bit length (001 - 000 1 0 - 001 instead of 001 - 010 - 010 - 010 - 001); these variations can be refined and would be described clearer and more detailed in further iterations.

A new dataset will likely be employed in the next iteration, and it should be filtered to remove the problems of non-words in the dataset. Additionally, vaster, more representative, and expandable sample texts need to be used to calculate the efficiency of the next iteration.

Conclusion

Efficiency-wise there was only a minor improvement in v2 over v1, but almost every semantic loss problem with v1 was addressed, except for undefined character encoding. Undefined words are whole and not atomized, uppercase characters are encoded uppercased, spaces can be in any quantity, and newlines and spaces are rendered according to the original input.

V2 does not reach much higher compression ratios than v1, and it is still on average only twice as efficient as ASCII, however, this is because v1 encodes less information than v2 does. V2 can be considered a success as it is not worse than v1 in efficiency, and it has large gains on encoded information and convenience of use over v1.

The demo webUI of the project is available on isef.heers.net/v2.

Third Iteration Write-Up

In the first two iterations, many of the parameters and functional elements were chosen arbitrarily and thus had large redundancies and inefficiencies built in. Parameters will be chosen more thoughtfully in this iteration.

Some ideas to increase efficiency are the following:

- Inferring the settings and parameters, such as dataset or indexing schemes, based on the output. The program can eliminate options that are impossible (binary length is invalid for certain parameters) and the receiver can choose the option that isn't non-sensical orderings of lexemes.
- There should be an option to reduce the size of the word model and use that to an advantage.
- Punctuation and letters should be swappable and arranged in statistically optimal ways.
- ID boundaries can be modified. Indexing should be more flexible and therefore optimizable.

Some other ideas for Iteration 3:

- Should be flexible and able to efficiently encode both simple and long messages.
- Datasets should be cleaner with non-words filtered out
- Support for multi-word lexemes (such as, as well as, be able to, etc.)

Cleaning and creating the web+ dataset for indexing scheme analysis

Figure 11 – Web+ dataset generation

```
1    !    2396
2    "    2773
4    $    108
5    %    8
6    &    1221
      ...
101  the  93174
102  and  61620
103  to   59680
104  of   49966
105  a    43361
106  in   32587
107  for  22839
108  is   22585
109  with 16808
110  on   15822
111  you  15132
112  that 14706
113  The  13769
114  are  13096
115  be   13039
      ...
```

Figure 11.1 – First entries in raw data

```
            ...
117577  ‡           1
117578  •For        1
117579  •Generate   1
117580  •Searching  1
117581  •Support    1
117582  •The        1
117583  •To         1
117584  ~4,000      1
117585  ≥38°C       1
117586  √           1
117587  ✓           1
117588  ▯           1
117589  ▯           1
117590  ﬁne         1
117591  ﬁnish       1
117592  ﬁrst        1
117593  Emma        1
117594  In          1
117595  The         1
117596  But         1
117597  &           1
```

Figure 11.2 – Last entries in raw data

```
     ...
1  'work
1  'works'
1  'world's
1  'worse
1  'worst
1  'worth
1  'would
1  'wow
1  'wrecked
1  'writers'
1  'written'
1  'yes
1  'yes'
1  'you'll
1  'you're
1  'zero
1  'zones'
1  ''do''
1  ''luckily
1  ''we
     ...
```

Figure 11.3 – Last entries after step 6

```
93174  the
61620  and
59680  to
49966  of
43361  a
32587  in
22839  for
22585  is
16808  with
15822  on
15132  you
14706  that
13769  the
13096  are
13039  be
11004  as
10419  your
10063  at
9572   have
9374   will
9236   from
    ...
```

Figure 11.4 – First entries after step 6

```
      ...
1 zestful
1 zests
1 zidovudine
1 zinc dust
1 zip up
1 zips
1 zir
1 zma
1 zodiac
1 zooarchaeology
1 zoology
1 zoom lens
1 zoomable
1 zooming
1 zooms
1 zoonoses
1 zoonosis
1 zorbing
1 zorbs
1 zu
1 zumba
```

Figure 11.5 – Last entries after step 9

To come up with a new indexing scheme, a new web+ dataset will be used. This one originates from the corpus, eng-uk_web-public_2018_10k-words, containing 117597 lexemes, available on the Deutscher Wortschatz from Universität Leipzig. The formatting of the raw data is shown in Figure 11.1. The first column is the entry number, the second is the lexeme, and the third is the occurrence of that lexeme in the eng-uk_web-public_2018_10k corpus. It is sorted by occurrence.

The plus in web+ means the dataset was "cleaned." Figure 11.2 demonstrates the quality of the "unclean" last entries. The following procedure was applied to clean the dataset:

1. Lowercase everything
2. regEx: `(?:^\d*?\t)(.*?)(?:\t)(\d*?$) => $2\t$1`
   a. First column is irrelevant
3. regEx: `‘|’ => '`

4. regEx: `−|—|‐|— => -`
5. Punctuation at the beginning is moved to a separate document for the next operations
6. regEx: `\d*?\t.*?[^a-z^'^-^\s].*?\r\n =>`
    a. Removes all entries containing characters other than letters, (`'`), (`-`), or `space`
    b. Removes `•To, ≥38°C`, etc.
7. regEx: `\d*?\t[^a-z].*?\r\n =>`
    a. Removes any entry beginning with a non-letter (Figure 11.3)
    b. Run twice to remove entries with two non-letters (`''we`)
8. regEx: `\d*?\t.*?[^a-z]\r\n =>`
    a. Removes any entry ending with a non-letter
9. Manually remove `1 ＆` on the last line
    a. Not an ampersand
    b. `＆`: Fullwidth Ampersand U+FF06
    c. `&`: Ampersand U+0026
10. regEx: `\d*?\t.\r\n =>`
    a. Removes single-letters from dataset (Figure 11.4)
11. Return punctuation to dataset
    a. Remove non-hyphen dashes, pointed quotes and apostrophes
12. Import data to a table in Excel
13. Remove duplicates in lexemes
14. Sort by occurrence
15. New cleaned dataset is 80782 lexemes long
16. The last 15247 rows are removed to make the dataset 65535 lexemes long

Deciding on an indexing scheme

Figure 12 – Proof of concept calculated manually, comparing three possible indexing schemes

```
Settings and Parameters         Results
Dataset: web+                   ASCII:      184
Indexing Scheme: Being Tested   NLC v2:     136
Punctuation & Letters at 32     Index @ 1.1 171
Letters after punctuation       Index @ 1.2 124
Uppercasing: True               Index @ 2.1 148

Original                 It   said      ,                    "          Hello                                           World     !      "
Lexeme      [uppercase] (it) (said)   [,]    [ ]    ["]    [uppercase] {hello}                             [uppercase] (world)   [!]    ["]
ID          [50]        (20) (144)    [33]   [49]   [42]   [50]        {59    51     62     62     77}    [50]        (266)     [43]   [42]
Binary      110010      10100 10010000 100001 110001 101010 110010      111011 110011 111110 111110 1001101 110010      100001010 101011 101010
Bit-Length  6           5    8        6      6      6      6           6      6      6      6      7       6           9         6      6

Index @ 1.1 10          5    10       10     10     10     10          10     10     10     10     10      10          10        10     10
Index @ 1.2 6           6    10       6      6      6      6           6      6      6      6      10      6           10        6      6
Index @ 2.1 7           5    10       7      7      7      7           7      7      7      7      7       7           10        7      7
```

Figure 12 details a possible procedure for encoding text to NLC. The sample text is `'It said, "Hello World!"'` because it includes uppercase letters, punctuation separated with space `', "'` and separated without `'!"'`. The test uses the web+ dataset, but with punctuation and letters starting at `lookupTable[32]`; letters after punctuation; and uppercasing (Figure 13 next page).

Three indexing schemes were tested: 1.1: [5, 10], 1.2: [6, 10], 2.1: [3, 5, 7, 10]

In the *Lexeme* step, the original text is split into its lexemes. Next, in the *ID* step, the lexemes are assigned their IDs according to the lookup table. In *Binary*, the IDs are converted to binary with no leading zeroes. *Bit-Length* shows the number of bits required for that ID and is used for determining the index-assignment.

The next lines are what the index-assignment of that ID would be for the different indexing schemes. The index-assignment is the smallest element of the indexing scheme that is ≥ the bitlength of the ID. For example, *said* becomes 144, becomes 10010000, which is 8 bits long. In indexing scheme 1.1, index-assignments can only be 5, or 10, and since 144 can only be expressed with ≥8 bits, the index assignment is 10. This adds 2 redundant bits. An ID that is 6 bits long would have 4 redundant bits in 1.1. These tests were carried out for each of the specified indexing schemes and the results for each are displayed in Figure 12 next to the settings and parameters.

Figure 13 – IDs 30-89 on lookupTable

30	our	60	d
31	their	61	l
32	.	62	f
33	,	63	c
34	-	64	m
35	'	65	u
36	\	66	g
37	$	67	y
38	(	68	p
39	)	69	w
40	:	70	b
41	?	71	v
42	"	72	k
43	!	73	j
44	&	74	x
45	;	75	z
46	€	76	q
47	*	77	0
48	<newline>	78	1
49	<space>	79	2
50	<uppercase>	80	3
51	e	81	4
52	t	82	5
53	a	83	6
54	o	84	7
55	n	85	8
56	r	86	9
57	i	87	<unknown>
58	s	88	not
59	h	89	which

Mathematically determining the most efficient indexing scheme for web+

It quickly became clear that testing every possible indexing scheme would be infeasible and there was no guarantee that its effectiveness for `'It said, "Hello World!"'` would be the same as for any other given text. Therefore, a more mathematical approach to the problem must be taken.

Added to the Excel sheet from pg. 32, were columns for: Rank, sorted first by occurrence and second by alphabetical order; Part of Text, the theoretical fraction of text that is this lexeme (lexeme's occurrence / sum of all lexemes' occurrences); and Sum Part of All, the sum of all fractions ranking greater than or equal to a lexeme (sum of all *Part of Text* cells above it). The first 16 and last four elements of this table are seen in Table 3.

Table 3 – Excel data table used for scatter plot

Rank	Lexeme	Length	Occurrence	Part of Text	Sum Part of All
1	the	3	93174	0.0525644	0.0525644
2	and	3	61620	0.0347631	0.0873276
3	to	2	59680	0.0336687	0.1209963
4	of	2	49966	0.0281885	0.1491848
5	in	2	32587	0.0183841	0.1675688
6	for	3	22839	0.0128847	0.1804535
7	is	2	22585	0.0127414	0.1931950
8	with	4	16808	0.0094823	0.2026772
9	on	2	15822	0.0089260	0.2116033
10	you	3	15132	0.0085368	0.2201401
11	that	4	14706	0.0082964	0.2284365
12	are	3	13096	0.0073882	0.2358247
13	be	2	13039	0.0073560	0.2431807
14	as	2	11004	0.0062079	0.2493886
15	your	4	10419	0.0058779	0.2552665
16	at	2	10063	0.0056771	0.2609436
…					
65533	aerosol	7	1	0.0000006	0.9999983
65534	afriyie	7	1	0.0000006	0.9999989
65535	ageaswe	7	1	0.0000006	0.9999994
65536	ageas's	7	1	0.0000006	1.0000000

A scatterplot, shown in Figure 14 (right), was created with the ranks as x-values and the occurrences as y-values. The axes were scaled logarithmically as base-2 for x and base-10 for y. A line was plotted from the first point to the last to show a rough line that the scatterplot follows. This pattern somewhat follows Zipf's Law, which states that the *n*-th most common word is inversely proportional to *n*. Figure 15 adds the Sum Part of All column on a secondary axis (right), which is scaled linearly from 0 to 1. It shows how much of the dataset consists of (y-axis) the first *n* words (x-axis). This has large implications as will be seen in Figure 16.

Figure 14 – Rank-Occurrence Scatterplot

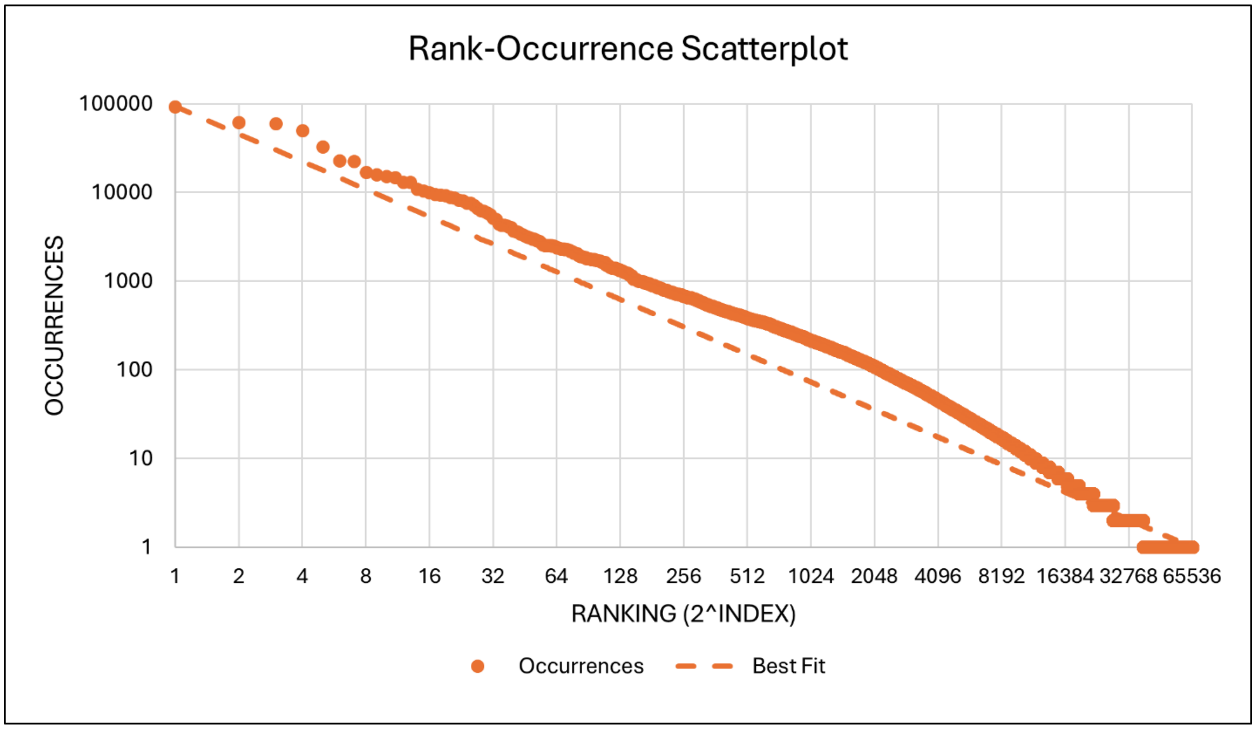


Figure 15 – Zipf Distribution on Web+

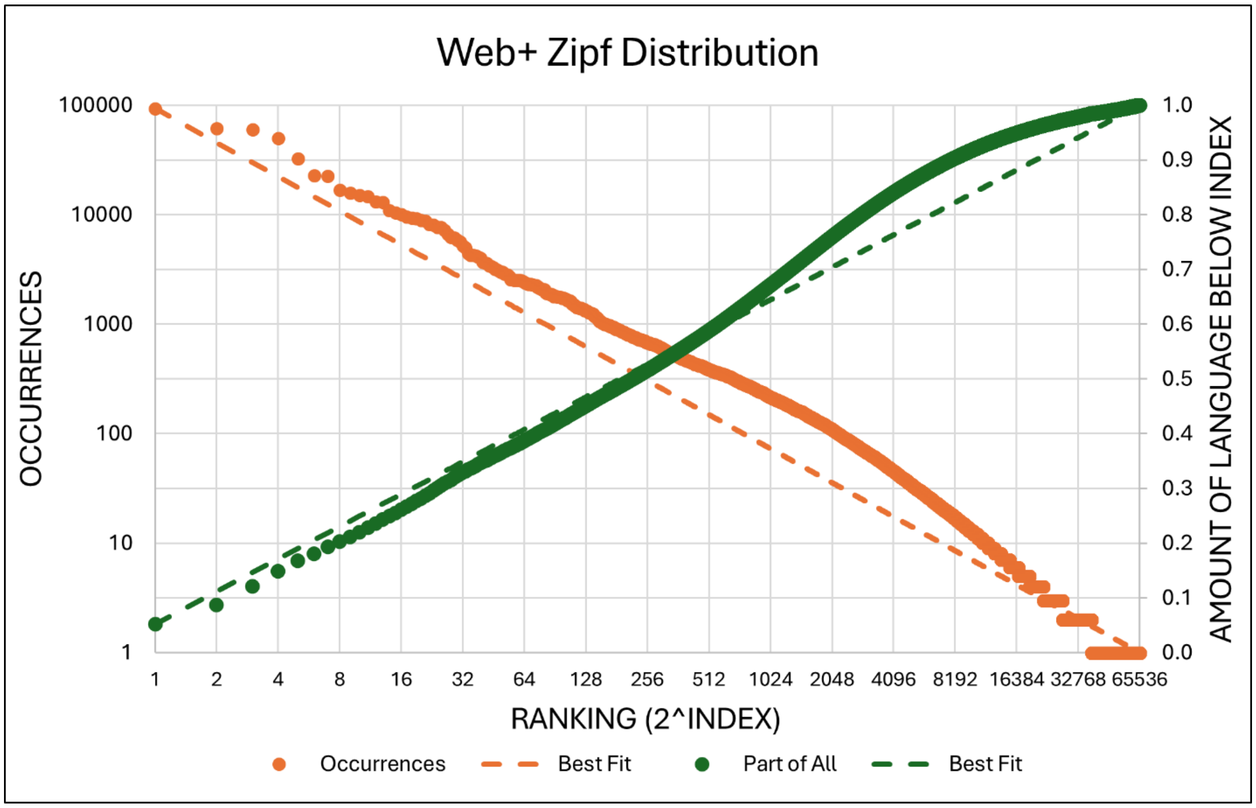
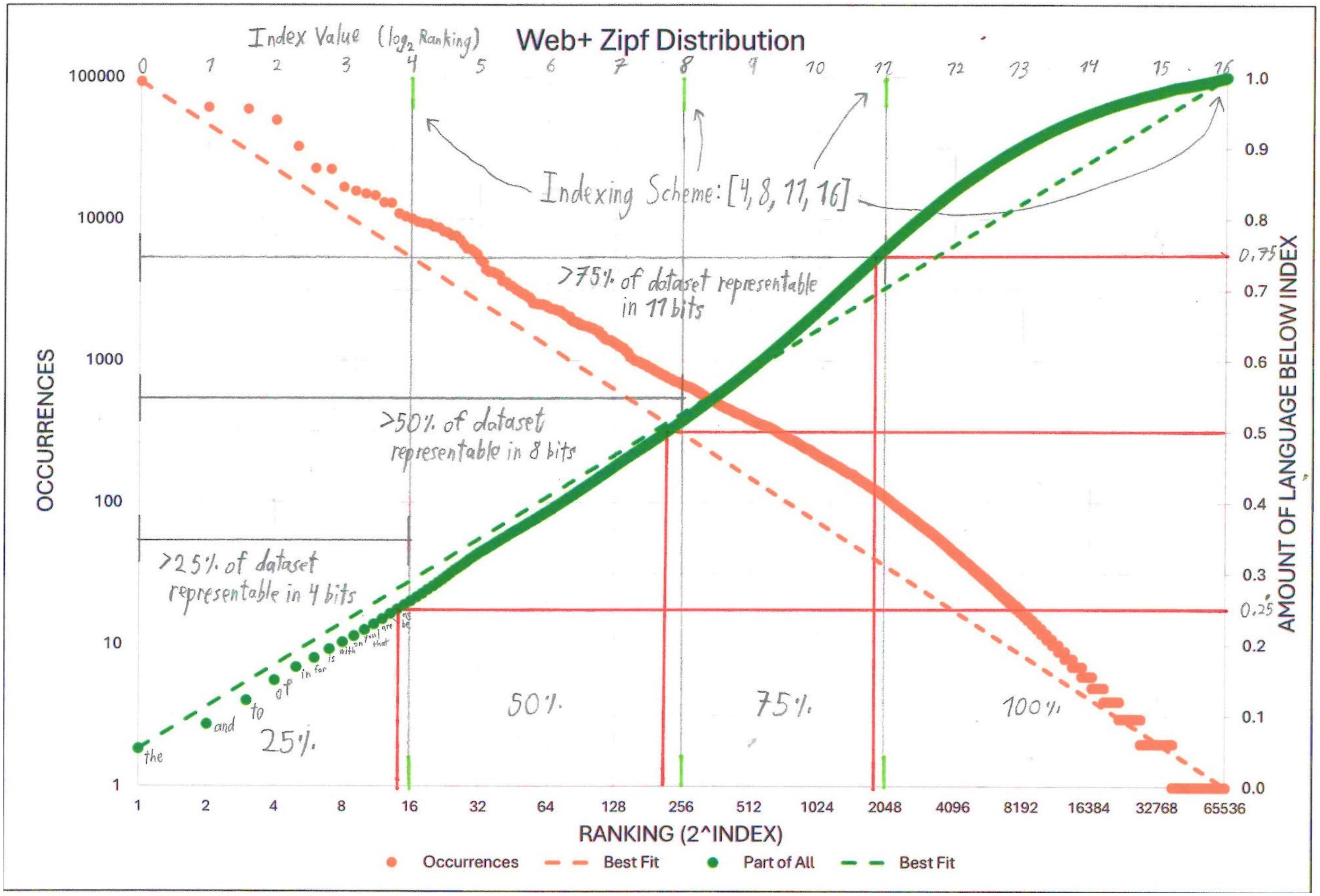


Figure 16 – Annotated Zipf Distribution Chart (Rotated Landscape 90° CCW)



Red lines are drawn from the quarter marks on the right y-axis (0.25, 0.5, 0.75) extending leftwards until intersecting with the green *Part of All* curve. After reaching the curve, the line extends downwards until intersecting with the x-axis. The next-highest index (top axis; $\log_2$ of *Ranking* (bottom axis)) to the right of the x-intercepts (light green marks) are added to the indexing scheme. This indexing scheme will then represent just over 25% of the dataset in 4 bits, 50% in 8, 75% in 11 and 100% in 16.

It is quite amazing that 25% of all English written natural language on the web is made up of only 14 words! However, while it is true that over 25% of the dataset is representable in 4 bits, those 16 words– *the, and, to, of, in, for, is, with, on, you, that, are, be, as, your, at* –cannot encode any meaningful sentences by themselves, and the remaining 75% is necessary for v3 to be at all practical.

This indexing scheme allows for 2-bit index assignments which can support 4 different (2^2) index assignments. These Excel spreadsheets and pencil sketches are proof of concept for how an algorithm should automatically be able to create an indexing scheme.

This algorithm should be able to create an indexing scheme for any given dataset, index assignment bit-length, desired index assignments (must be at most 2^index; can be less for special-case index assignments, as described in v2 Variations (Page 20)), and highest index. For example, this proof of concept used the web+ dataset, an index assignment bit-length of 2, 4 desired index assignments, and a highest index of 16.

Implementation

V3 is very different from v2 in functionality and the best course of action to keep the code clean and readable would be to restart and delete as much of it as possible. In subiteration 1 (S1), almost everything from v2 was deleted, except for some constants that may still be applicable, the binary functions, most of the HTML, and the functions facilitating the HTML.

S2 implements an algorithm to automatically carry out the function of creating an indexing scheme as detailed in Figure 16. Two variables are used to customize the configuration of an indexing scheme. highestIndex is the maximum bit-length a binary ID can be and caps the elements a dataset can contain at 2 to the power of its value. desiredIndexes is the number of indexes that are desired in the indexing scheme. This should be a power of two or slightly less than a power of two: S2's example has it set to 3 so that the index assignments are 2 bits long but a special-case 11 index assignment could be programmed in.

Dataset Setup

A different version of the web+ dataset will also be used for the program itself. Its setup process is simpler, does not require the use of Excel, and could be used for different dataset sources if necessary. Currently, the setup process uses RegEx operations in Notepad++, but it could be configured to run those operations as part of the JavaScript, allowing custom corpus use to be more streamlined. The following is the procedure for configuring the dataset:

1. Download eng-uk_web-public_2018_100K-words.txt from Wortschatz Leipzig[6]

[6] https://wortschatz.uni-leipzig.de/de/download/English web-public United Kingdom 100K

2. The following find and replace operations are performed in Notepad++ with RegEx active and “. matches newline” selected
3. *Replace all* (^\d*?\t)(.*?)(\t)(\d*?)($) *with* $4\t$2
4. Sort lines as integers descending
5. Count [A-Z] occurrences (there should be 90007 uppercase characters) (make sure Match Case is active)
6. Insert ␑ with 90007 occurrences as an uppercase control character at its appropriate location (keeping everything sorted as above and formatted the same as the other entries)
7. Copy all and paste to a new document named x16web.js
8. In x16web.js:
    a. Remove every element with an occurrence of one: *Replace all* ^1\t.*$ *with nothing*
    b. Copy all and paste to a new temporary document
9. In x16web.js:
    a. Remove every element with an occurrence of one: *Replace all* ^\d*?\t *with nothing*
    b. *Replace all* ^.*?$ *with* \t"$0",
    c. Declare it as an array of x16webWords: *Replace* .* *with* const x16webWords = [\n$0\n]

   d. Delete the empty string at the bottom: *Replace* \t"",\n *with nothing*

10. In the temporary document created in step 8.b:

   a. *Replace all* (^\d*?)(\t.*?$) *with* \t"$1",

   b. *Replace* .* *with* const x16webOccurrences = [\n$0\n]

   c. Delete blank line at bottom: *Replace* \n\n *with nothing*

   d. Copy all and paste at the end of x16web.js

11. Save the original document (everything before step 7) as web+.txt

12. Save the x16web.js document

13. Delete the temporary document/tab

Figure 17 – Dataset before and after operations

Before						After	
1	!	2396	101	the	93174	const x16webWords = [	const x16webOccurrences = [
2	"	2773	102	and	61620	"the",	"93174",
4	$	108	103	to	59680	".",	"90504",
5	%	8	104	of	49966	"␑",	"90007",
6	&	1221	105	a	43361	",",	"74919",
	…			…		…	…
85	–	2680	117593	Emma	1	"0-1",	"2",
86	—	110	117594	In	1	"0-5",	"2",
87	−	3	117595	The	1	"0-16",	"2",
88	―	2	117596	But	1	"0-25",	"2",
90	·	16	117597	&	1	]	]

In Figure 17, each side's right column is a direct continuation of its left column and ellipses show omitted lines: both sides go top to bottom, left to right with the vertical dividers representing a newline. Additional *sentence* datasets for later use have their setup detailed in the following steps:

1. Download eng-uk_web-public_2018_300K-sentences.txt from Wortschatz Leipzig[7]
2. *Replace all* ^\d*?\t *with nothing*
3. *Replace all* " *with* \\"
4. *Replace all* ^.*?$ *with* \t"$0",
5. *Replace all* " *with* \\"
6. Declare it as an array of sampleSentences: *Replace* .* *with* const sampleSentences = [\n$0\n]
7. Save it as eng-uk_web-public_2018_300K-sentences.js
8. Copy all and paste to a new document; in the new document:
    a. Remove everything after the 10001st line
    b. Type] into the 10002nd line
    c. Save this new document as webSampleSentences10k.js

Indexing Scheme Calculation (Subiteration 2 & 3)

Currently the dataset has 50421 elements, whose $\log_2$ is approximately 15.62. This means it takes 16 bits to be able to identify each element (hence the name x16web.js), but 46% of that

[7] https://wortschatz.uni-leipzig.de/de/download/English web-public United Kingdom 300K

16th bit will be unused and wasted ($\frac{2^{16}-50421}{2^{16}-2^{15}}$) (empty space in the 16th bit divided by the space the 16th bits adds on top of the 15th). Therefore, it is recommended to trim the dataset to at least 15 bits. Trimming is set in the highestIndex variable, which slices the words and occurrences arrays from 0 to 2^highestIndex.

To create an analog to the Sum Part of All column in Table 3 on page 35, the program generates the percentOfWhole array using the occurrences array. It then generates the first index in the indexing scheme by iterating through percentOfWhole until a value that is greater than or equal to 1/desiredIndexes is reached; this is equivalent to drawing the horizontal red lines in Figure 16 from each green dot until one of them gets over the 25% mark (1/4, when desiredIndexes was 4). Once that specific index is reached, the $\log_2$ of the *i* (loop indices will be expressed as *i* to avoid confusion with NLC indexes) at that percentOfWhole value (the vertical red line in Figure 16) is rounded and pushed to the indexing scheme. The numerator (from *1/desiredIndexes*) is then increased by one and the process is repeated until all desired indexes have been added to the indexing scheme.

S3 refactors this process into the function, calculateIndexingScheme, taking highestIndex, desiredIndexes, indexAssignmentBitLength, and dataset as parameters. Dataset is an object in the form of `{lexemes: [], occurrences: []}`. x16web.js is also refactored to an object `x16web = {lexemes: [...], occurrences: [...]}`, but maintaining the same values as configured previously.

An additional function added in S3 is "simulating" the dataset to get a rough idea of the efficiency of different indexing schemes. The simulation is done by iterating over each lexeme in the dataset and determining the smallest index that 2 can be raised to that is greater than the *i* of the iteration. This essentially assumes *i* to be the ID of that lexeme.

A counter for totalBitsNLC is increased by the (smallest index[8] + indexAssignme ntBitLength) * the occurrence of that lexeme. A counter for totalBitsASCII is also increased, but by the length of that lexeme * 8 (each ASCII character is 8 bits) * the occurrence of that lexeme. The function, calculateIndexingScheme, returns the original dataset, totalBitsASCII, totalBitsNLC, and the ratio between the two former.

Indexing Scheme Analysis (S3 & S4)

Now that there is a quick, automatic way to generate indexing schemes with different parameters, the next step is to analyze the different possible indexing schemes and decide on the one to use for further implementation. The parameters were iterated from 15 to 1 for highestIndex, 1 to 3 for indexAssignmentBitLength, and 2^indexAssignmentBitLength and 2^indexAssign-mentBitLength for desiredIndexes. The results of the test are shown in Table 4 (next page).

Only a few of these options are at all practical for this project and because an 8-bit highest index is too small to encode natural language, the main decision is to be made is whether to use a 15-bit highest index with an indexAssignmentBitLength of 3 or 2. To allow flexibility for a planned use of 11 or 111 as a special index, the best choice and the one that will be used for this project, is the highlighted yellow row.

S4 cleans everything up into a data analysis section on the HTML. It also redefines the term for creating an indexing scheme to *configuring* the dataset. Configuring the dataset means

[8] The exact terminology was not finalized at the time S3 was written; smallestIndexAssignment refers to the index at the smallest index assignment

taking in a raw dataset (object in the form of {lexemes: [], occurrences: []}), and returning an object containing the "simulation" and the indexingScheme ({...dataset, totalBitsASCII, totalBitsNLC, simulatedRatio, indexingScheme}.) In order to avoid doing this calculation every time the program is refreshed or run, a preconfigured web-15.js dataset was created by taking the returned object from datasetConfig with the parameters being highestIndex=15, desiredIndexes=7, indexAssignmentBitLength=3, dataset=x16webRaw, and declaring it as an object, let web15 = {…}.

Table 4 – Results of the indexing scheme analysis

highest Index	desired Indexes	indexAssignment BitLength	ASCII Bits	NLC Bits	Ratio	Indexing Scheme
15	2	1	80200720	26923584	2.979	7,15
15	1	1	80200720	36087488	2.222	15
15	4	2	80200720	24457032	3.279	3,7,11,15
15	3	2	80200720	26205915	3.060	4,9,15
15	8	3	80200720	24694061	3.248	2,3,5,7,9,11,12,15
15	7	3	80200720	25016382	3.206	2,3,6,8,10,12,15
14	2	1	75247264	24663374	3.051	7,14
14	1	1	75247264	32681790	2.302	14
14	4	2	75247264	22817576	3.298	3,7,10,14
14	3	2	75247264	24148476	3.116	4,9,14
14	8	3	75247264	23057501	3.263	1,3,5,7,9,10,12,14
14	7	3	75247264	23360941	3.221	2,3,5,8,10,12,14
13	2	1	68656248	21949949	3.128	6,13,13
13	1	1	68656248	28991746	2.368	13,13
13	4	2	68656248	20583799	3.335	3,6,10,13,13
13	3	2	68656248	21775426	3.153	4,9,13,13
13	8	3	68656248	21078084	3.257	1,3,4,6,8,10,11,13,13
13	7	3	68656248	21257370	3.230	2,3,5,7,9,11,13,13
…						
8	2	1	27300488	8721512	3.130	3,8
8	1	1	27300488	11537667	2.366	8
8	4	2	27300488	8769391	3.113	2,3,6,8
8	3	2	27300488	9171909	2.977	2,5,8
8	8	3	27300488	9421417	2.898	0,2,3,3,5,6,7,8
8	7	3	27300488	9405837	2.903	0,2,3,4,5,7,8
…						
2	2	1	4279616	859464	4.979	0,2,2
2	1	1	4279616	1045812	4.092	2,2
2	4	2	4279616	1117564	3.829	-Infinity,0,1,2,2
2	3	2	4279616	1117564	3.829	0,1,2,2
2	8	3	4279616	1466168	2.919	-Infinity,0,1,2,2
2	7	3	4279616	1466168	2.919	-Infinity,0,1,2,2
1	2	1	2960208	274182	10.797	-Infinity,0,1
1	1	1	2960208	274182	10.797	0,1
1	4	2	2960208	457860	6.465	-Infinity,0,1
1	3	2	2960208	457860	6.465	-Infinity,0,1
1	8	3	2960208	641538	4.614	-Infinity,0,1
1	7	3	2960208	641538	4.614	-Infinity,0,1

Lexeme Determination by Highest-Ratio Substringing (LDHR) (S5 & S6)

In v2, splitting the input string into IDs by words was relatively simple and easy to implement. However, one of the goals designated in the Engineering Plan (pg. 6) was the implementation of lexemes beyond just a word. A lexeme is an abstract unit of written language: it can be a word, multiple words, a phrase, or any sequence of letters that can be used together. Conveniently, Wortschatz Leipzig's corpora define their "words" similarly to how lexemes are defined here, and they also have all their lexemes sorted by occurrence. Some common lexemes consisting of more than one word in the x16web dataset are: (on the), (need to), (such as), (up to), (as well as), (in order to), (as a result of), (at the same time), (in the same way as).

In S5, the dataset configuration creates an array parallel to lexemes and occurrences called ratios, the ratio of each lexeme's ASCII encoding to its NLC encoding. These three arrays are then sorted in parallel by their ratios. This puts the top element in the dataset (the one with the highest ratio) as (Financial Services Compensation Scheme) - 16.89 times as efficient than ASCII, followed by (Area of Outstanding Natural Beauty) - 15.11, and (Information Commissioner's Office) - 14.67.

Lexeme Determination by Highest Ratio Substringing (LDHR) is the process of splitting an input string by progressively iterating from the highest ratio lexemes to the lowest and splitting the substrings into an array of lexemes. LDHR is visualized in the following Figure 18.

In Figure 18, the process is visualized with the sample input `'it said, "hello, the world!"'`. The whole input starts off as unencoded (red), and the dataset is iterated from the highest ratio descending, until a lexeme that is a substring in the input is reached ((the world) - 5.54). That substring becomes encoded (green) and splits the unencoded in two on either side. This is repeated until all the input is encoded. This is more efficient than splitting by words: (the world) takes up 13 bits (100 - 0110111000) compared to (the) (world) taking up 16 (000 011 - 00 | 11111010).

Figure 18 – LDHR Process Visualization

```
     it said, "hello, the world!" [it said, "hello, the world!"]
5.54                  the world   [it said, "hello,] (the world)[!"]
2.91    said                      [it] (said)[, "hello,] (the world)[!"]
2.22           hello              [it] (said)[, "](hello)[,] (the world)[!"]
1.78 it                           (it) (said)[, "](hello)[,] (the world)[!"]
1.60        ,       ,             (it) (said)(,) ["](hello)(,) (the world)[!"]
0.73          "                 " (it) (said)(,) (")(hello)(,) (the world)[!](")
0.73                           !  (it) (said)(,) (")(hello)(,) (the world)(!)(")
```

This usually works as the most efficient method of splitting– unless a high-ratio lexeme is only a part of a longer lexeme. For example, suppose "tone" is being encoded. (tone) is encoded in 18 bits and has a ratio of 1.78, but (to) has a ratio of 2.76, so in the first step, [tone] will be split to (to)[ne], and then (to)(ne), which is encoded in 24 bits (001 110 - 101 | 110000111100100), and (ne) alone is 18 bits, which was as much as (tone).

S6 provides a partial solution to the problem of words being split apart by higher-ratio lexemes. To keep lexemes from splitting inefficiently, the split lexemes are passed through an optimizer that tests if LDHR’s splitting is less efficient than encoding the lexeme as a single word. If the word can be encoded as a lexeme, it is, but if it cannot, the optimizer will split the word into every possible configuration of up to three lexemes and pick the most efficient one. (The latter is not actually implemented until S10)

Using this method, [tone] will first be split to (to)[ne], and then the optimizer will test if (tone) exists and if it is more efficient. Since it does exist, it will mark tone to be split by itself as it passes everything back to the splitIntoArray function.

However, if an inefficiently split word is not a lexeme in the dataset itself, like [seeker], which is split in LDHR as (see)(k)(e)(r), and when checked if (seeker) is a lexeme, it is not, then each possibility of splitting into up to 3 lexemes is tested (Figure 19). The most efficient of these splits is (seek)(er), and that becomes the chosen split.

Figure 19 – Optimization Process

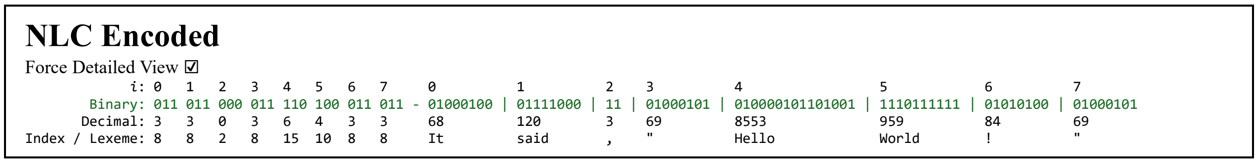
s, eeker	seek, er	s, eek, er	seek, er
se, eker	seeke, r	se, eke, r	seeke, r
see, ker	seek, er	see, ker	s, eeker
seek, er	s, eeke, r	seek, er	s, e, eker
seeke, r	se, eker	seeke, r	s, e, eker
seeke, r	see, ker	se, eker	see, k, er
s, eeker	seek, er	s, ee, ker	s, e, eker
se, eker	seeke, r	se, ek, er	seeke, r
see, ker	see, ker	see, ke, r	

Enhanced Bracketing and Abstraction Inversion (S7 & S8)

S7 implements the same bracketing visualization that v2 has when “Render Brackets” is selected. Lexemes are green and enclosed in parentheses, while spaces are yellow and enclosed in square brackets, but tt is important to note that spacing is not yet implemented in the encoding process. When brackets are enabled, they also display in the binary in the NLC Encoded section, where index assignments are split by spaces, the index is separated from the content with a hyphen, and IDs are split by pipes. ASCII is also split by spaces for each character (each byte).

S8 provides enhanced debugging, abstraction inversion (opposite of abstraction, shows more details on the lower-level processes to the user), and therefore, a more detailed visualization of how the natural language is being encoded and how to optimize inputs.

Figure 20 – Detailed Bracket View (Adjusted from dark-mode)



NLC Encoded

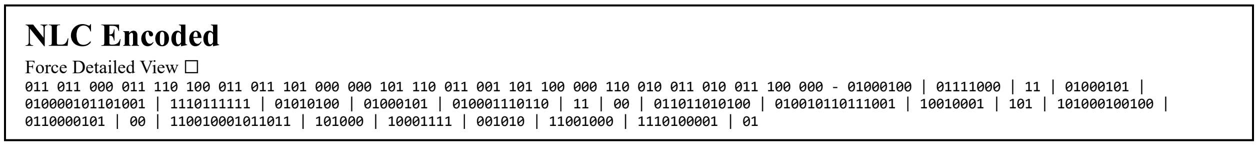
Force Detailed View ☑

```
             i: 0   1   2   3   4   5   6   7     0          1          2    3          4                 5            6          7
        Binary: 011 011 000 011 110 100 011 011 - 01000100 | 01111000 | 11 | 01000101 | 010000101101001 | 1110111111 | 01010100 | 01000101
       Decimal: 3   3   0   3   6   4   3   3     68         120        3    69         8553              959          84         69
Index / Lexeme: 8   8   2   8   15  10  8   8     It         said       ,    "          Hello             World        !          "
```

Figure 20 shows what is displayed, as an input is typed into the input field. *i* is the index (not NLC index) of each lexeme, so that IDs can be associated to their index assignments. Binary is what is shown normally in S7 bracketing. Decimal is the decimal value of each binary value above it. Index / Lexeme is the interpretation or meaning of that decimal/binary value. At the *i* of 0, an index assignment of 011 means the index is element 3 of indexingScheme ([2, 3, 6, 8, 10, 12, 15]), which is 8. This means that the ID *i* of 0 is 8 bits long. Because (It)'s ID is 68, or 1000100 in binary (7 bits), it adds a leading zero to match with the index.

When the input is too long, and the detailed view cannot be shown (because of line wrap), it reverts to the S7 visual (Figure 21), however, the detailed view can be forced to show (in case one wishes to copy it for another program, or if it can just barely fit onto the screen).

Figure 21 – S8 Fallback Bracket View (Adjusted from dark-mode)



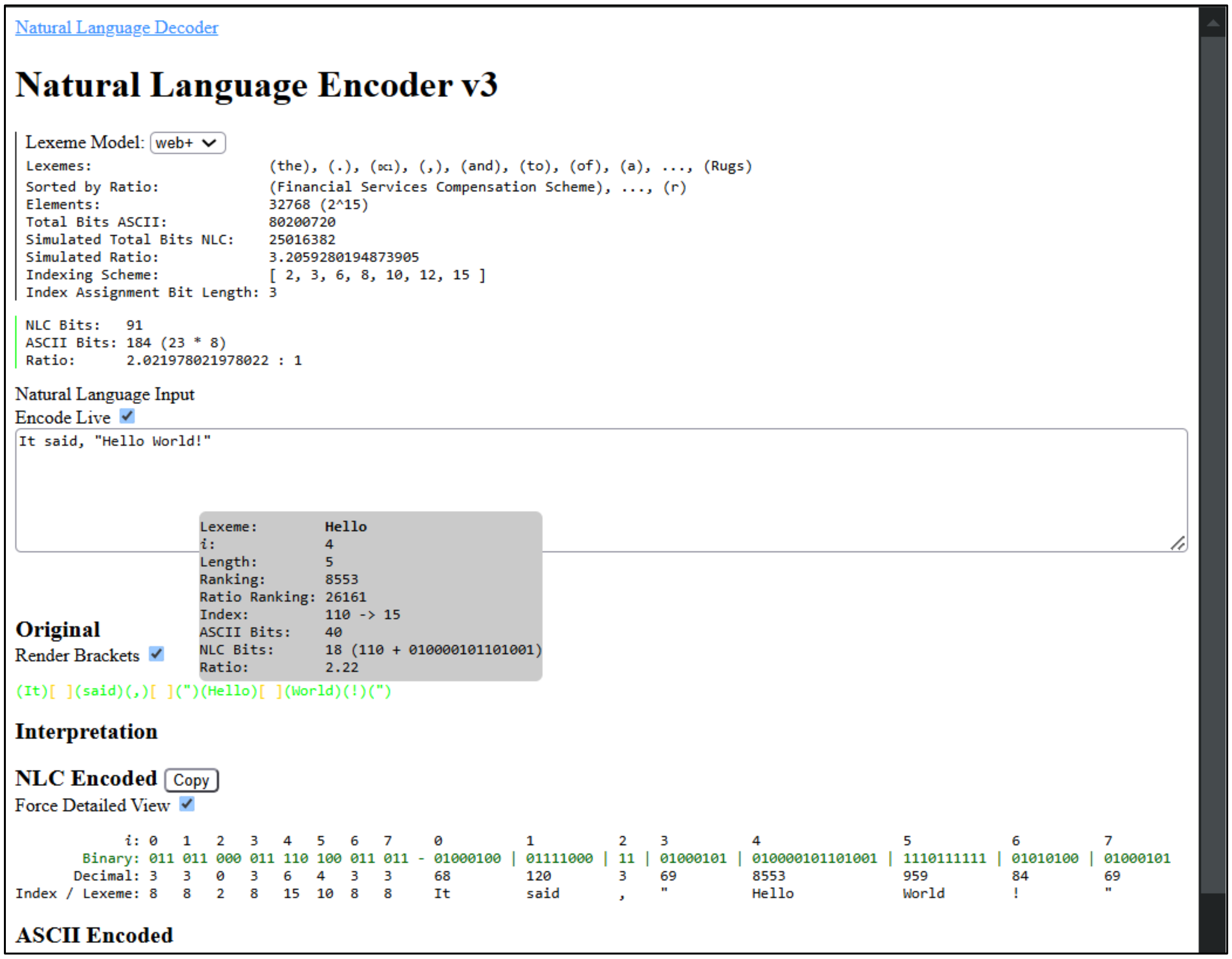
NLC Encoded

Force Detailed View ☐

```
011 011 000 011 110 100 011 011 101 000 000 101 110 011 001 101 100 000 110 010 011 010 011 100 000 - 01000100 | 01111000 | 11 | 01000101 |
010000101101001 | 1110111111 | 01010100 | 01000101 | 010001110110 | 11 | 00 | 011011010100 | 010010110111001 | 10010001 | 101 | 101000100100 |
0110000101 | 00 | 110010001011011 | 101000 | 10001111 | 001010 | 11001000 | 1110100001 | 01
```

Tooltips are also an addition to S8, where the user can hover their mouse over a lexeme in the Original section (view Figure 24 for the webUI on Page 55) and view information about it. This feature is shown in Figure 22 (next page).

Figure 22 – S8 Tooltip Screenshot



Natural Language Decoder

Natural Language Encoder v3

Lexeme Model: web+

```
Lexemes:                     (the), (.), (␑), (,), (and), (to), (of), (a), ..., (Rugs)
Sorted by Ratio:             (Financial Services Compensation Scheme), ..., (r)
Elements:                    32768 (2^15)
Total Bits ASCII:            80200720
Simulated Total Bits NLC:    25016382
Simulated Ratio:             3.2059280194873905
Indexing Scheme:             [ 2, 3, 6, 8, 10, 12, 15 ]
Index Assignment Bit Length: 3
```

```
NLC Bits:   91
ASCII Bits: 184 (23 * 8)
Ratio:      2.021978021978022 : 1
```

Natural Language Input

Encode Live ✓

```
It said, "Hello World!"
```

```
Lexeme:        Hello
i:             4
Length:        5
Ranking:       8553
Ratio Ranking: 26161
Index:         110 -> 15
ASCII Bits:    40
NLC Bits:      18 (110 + 010000101101001)
Ratio:         2.22
```

Original

Render Brackets ✓

```
(It)[ ](said)(,)[ ](")(Hello)[ ](World)(!)(")
```

Interpretation

NLC Encoded Copy

Force Detailed View ✓

```
             i: 0   1   2   3   4   5   6   7     0          1          2    3          4                 5            6          7
        Binary: 011 011 000 011 110 100 011 011 - 01000100 | 01111000 | 11 | 01000101 | 010000101101001 | 1110111111 | 01010100 | 01000101
       Decimal: 3   3   0   3   6   4   3   3     68         120        3    69         8553              959          84         69
Index / Lexeme: 8   8   2   8   15  10  8   8     It         said       ,    "          Hello             World        !          "
```

ASCII Encoded

In Figure 22, the cursor is hovering over the lexeme (Hello). *i* is used the same as it is used in the detailed view; length is how many characters long it is; ranking is its rank in order of frequency in the dataset; ratio ranking is the rank in order of ratios; index shows the index assignment, followed by what index that index assignment translates to; ASCII and NLC bits show how many bits it takes to encode that word, and NLC bits also includes what those bits are as an index assignment and ID; and ratio shows the ratio between ASCII and NLC bits.

Major Bug Fixes and Prototype Release (S9 & S10)

S9 is a minor subiteration replacing unencodable lexemes with the SUB character, previously known as the unknown character (�). The last elements of x16web.js were temporarily changed to have lexemes 32741–32768 replaced with the simple Latin alphabet and the SUB character. This prevents the program from crashing if there is an unencodable lexeme.

S10 fixed the largest bugs, which will likely make this project worthy of being a prototype release. Among the bugs were:

- optimizeArray could not optimize any lexemes that were in contact with a punctuation, for example: "it.". This was fixed by testing every single possible substring inside of a word, splitting by it, and choosing the configuration that is the most efficient. (Page 48)
- "edit it." would become “\ed\it\\\it.\” in optimizeArray because the "." in "it." would evaluate to ([^\\]||$)(it.)([^\\]||^) in the RegEx, where the "." would mean a wildcard character. The problem was solved by changing the RegEx line to wordArray = wordArray.join(" ").replaceAll(eval(`/([^\\\\]||$)(${testedWord.replaceAll(/([\^\$\\\.*\+\?\(\)\[\]\{\}\|\/])/g, "\\$1")})([^\\\\]||^)/g`), "$1\\$2\\$3").split(" ")
- When two lexemes that are broken up in LDHR and put back together are the same ("edit the edit", for example), the program would fail to encode it. The problem was caused by a misconfiguration in the RegEx, and solved by changing the line to wordArray.join(" ").replaceAll(eval(`/([^\\\\]{1}|^)(${testedWord.replaceAll(/([\^\$\\\.\*\+\?\(\)\[\]\{\}\|\/])/g, "\\$1")})([^\\\\]{1}|$)/g`), "$1\\$2\\$3").split(" ")

S10 Spacing Handling

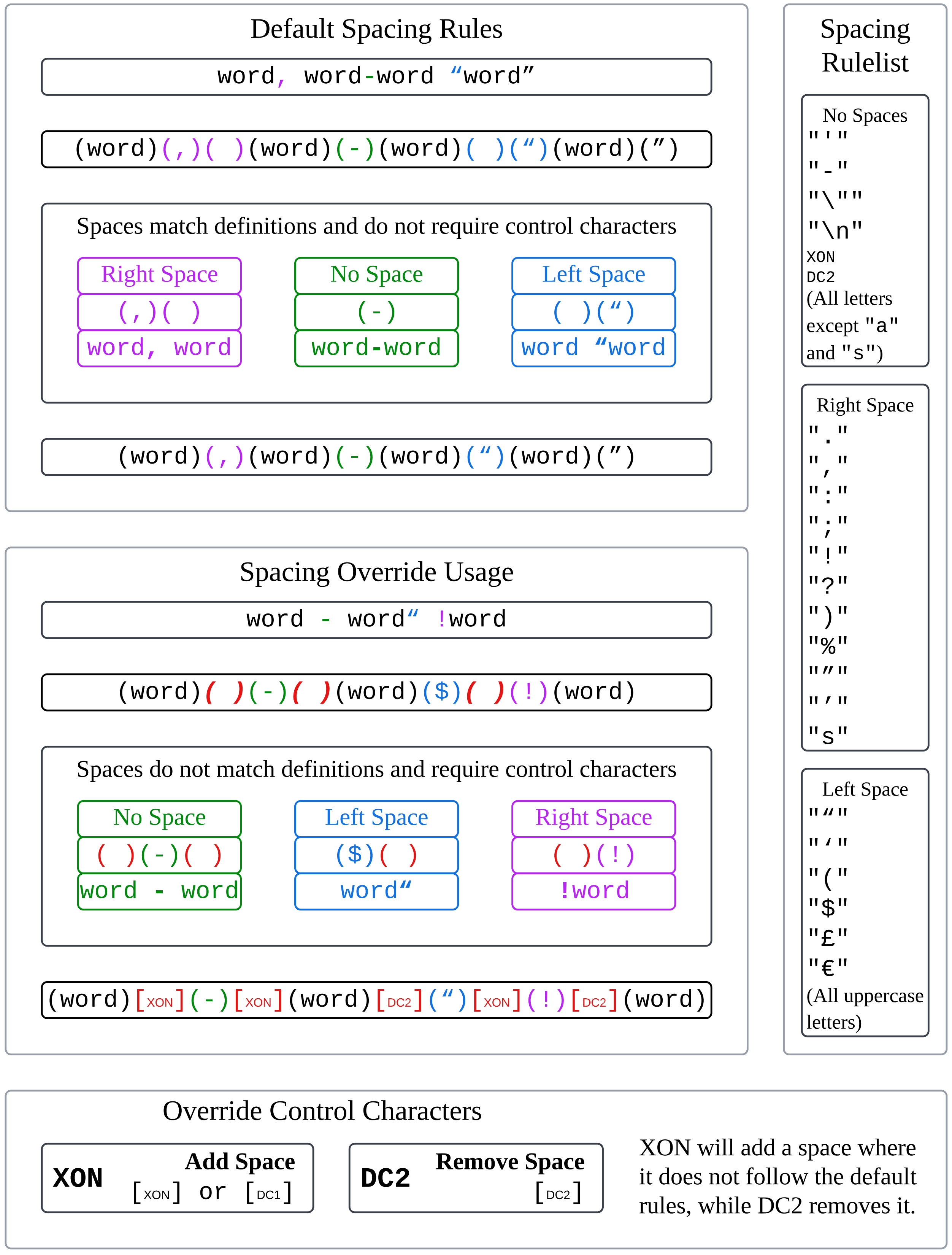
Since lexemes can contain spaces, and some words are made of multiple lexemes not separated by spaces ((seek)(er)), a new spacing handling method must be implemented.

Many characters have regular uses of spaces around them. For example, a period and comma are usually followed by a space only on its right (This is a sentence. This is another sentence.) An apostrophe usually has no spaces (can't, John's), and a dollar-sign usually has a space only on its left (It costs $20.)

By default, every lexeme has a space on its left, but some characters have special rules defined in the spacing rulelist on the right (Figure 23). In the sample text `"word, word-word “word”"` (note the curly quotes), all of the punctuation marks and lexemes follow the default spacing rules and do not require any extra encoding for the spaces, i.e. it is encoded as (word)(,)(word)(-)(word)(“)(word)(”), with no space markers.

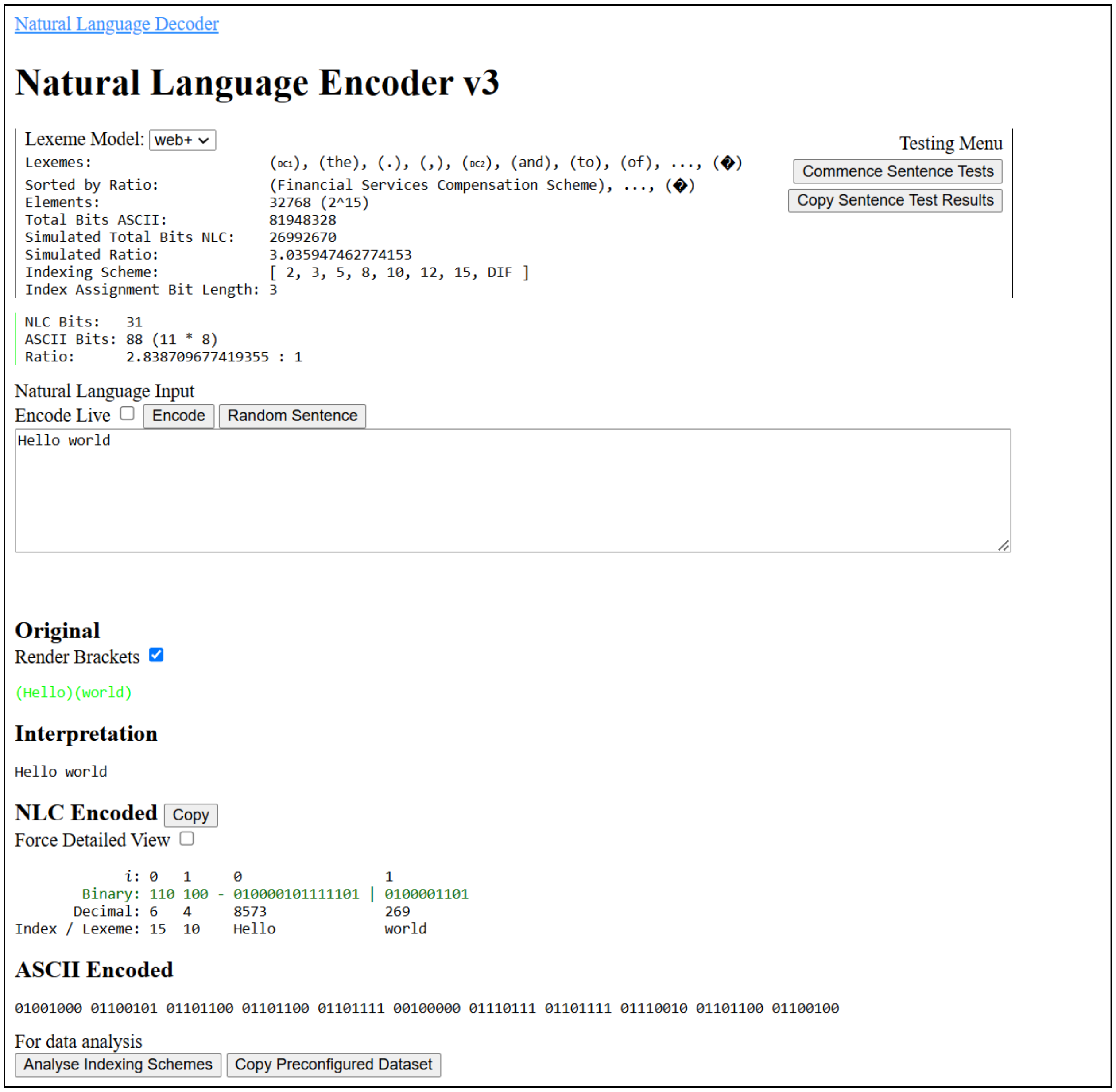
If however, there is a non-default spacing usage of a character, as in the sample text `"word – word$ !word"`, the use of spacing override characters is required. Because lexemes are now either uppercased or lowercased, the XON control character is used to add a space where there otherwise would not be one, and the DC2 control character is used to remove a space where there would usually be one.

Figure 23 – v3 Spacing Handling


Default Spacing Rules
word, word-word “word”
(word)(,)( )(word)(-)(word)( )(“)(word)(”)
Spaces match definitions and do not require control characters
Right Space
(,)( )
word, word
No Space
(-)
word-word
Left Space
( )(“)
word “word
(word)(,)(word)(-)(word)(“)(word)(”)
Spacing Override Usage
word - word“ !word
(word)( )(-)( )(word)($)( )(!)(word)
Spaces do not match definitions and require control characters
No Space
( )(-)( )
word - word
Left Space
($)( )
word“
Right Space
( )(!)
!word
(word)[XON](-)[XON](word)[DC2](“)[XON](!)[DC2](word)
Spacing Rulelist
No Spaces
"'"
"-"
"\""
"\n"
XON
DC2
(All letters except "a" and "s")
Right Space
"."
","
":"
";"
"!"
"?"
")"
"%"
"”"
"’"
"s"
Left Space
"“"
"‘"
"("
"$"
"£"
"€"
(All uppercase letters)
Override Control Characters
XON
Add Space
[XON] or [DC1]
DC2
Remove Space
[DC2]
XON will add a space where it does not follow the default rules, while DC2 removes it.

Web UI Guide

Figure 24 – UI for the Natural Language Encoder


Natural Language Decoder

Natural Language Encoder v3

Lexeme Model: web+

```
Lexemes:                    (DC1), (the), (.), (,), (DC2), (and), (to), (of), ..., (�)
Sorted by Ratio:            (Financial Services Compensation Scheme), ..., (�)
Elements:                   32768 (2^15)
Total Bits ASCII:           81948328
Simulated Total Bits NLC:   26992670
Simulated Ratio:            3.035947462774153
Indexing Scheme:            [ 2, 3, 5, 8, 10, 12, 15, DIF ]
Index Assignment Bit Length: 3
```

Testing Menu

```
NLC Bits:   31
ASCII Bits: 88 (11 * 8)
Ratio:      2.838709677419355 : 1
```

Natural Language Input

Encode Live ☐

```
Hello world
```

Original

Render Brackets ☑

```
(Hello)(world)
```

Interpretation

```
Hello world
```

NLC Encoded

Force Detailed View ☐

```
             i: 0   1     0                1
        Binary: 110 100 - 010000101111101 | 0100001101
       Decimal: 6   4     8573             269
Index / Lexeme: 15  10    Hello            world
```

ASCII Encoded

```
01001000 01100101 01101100 01101100 01101111 00100000 01110111 01101111 01110010 01101100 01100100
```

For data analysis

In Figure 24, the dataset menu is the left half of the segment with the left black border. You can select the lexeme model (once multiple models are implemented) and view statistics of that dataset. Lexemes shows the first 8 lexemes and the last lexeme in order of frequency (what they would be indexed as); Sorted by Ratio shows the most efficient lexeme and the least efficient lexeme; Elements shows how many elements are in the dataset, along with the power of two it is; Total Bits ASCII, Simulated Total Bits NLC, and Simulated Ratio are the simulated values from the dataset configuration process; Indexing Scheme is the indexing scheme of the dataset; and Index Assignment Bit Length is the amount of bits that are required for each index assignment.

The statistics group is the area with the left green border. It shows the actual statistics of the text being encoded: NLC bits, ASCII bits, and the ratio between the two.

The input is typed into the textarea where the user can choose to have the input be encoded live as it is typed, or to only encode once the encode button is pressed (recommended for performance). The Random Sentence button will autofill the textarea with a randomly selected sentence from the webSentences dataset (either 10k or 300k).

Original will show the lexeme boundaries of the input, and where spacing override characters are placed. If bracket rendering is turned off, it will not show the lexeme boundaries.

Interpretation runs the decode function on the output to get the actual output that will be received when the text is decoded.

NLC Encoded will show the binary output, and if brackets are activated, it will show the index and ID boundaries or a detailed view, if there is enough space.

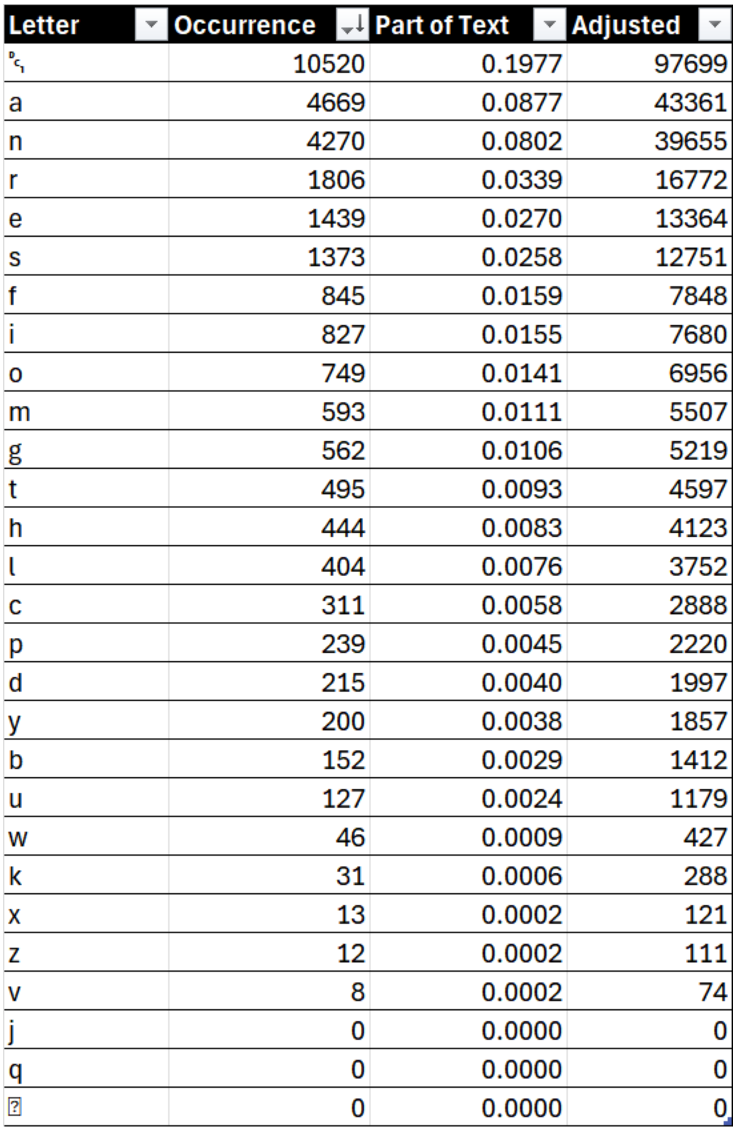
The Data Analysis section is used to preconfigure the dataset and create the table for indexing scheme analysis (page 46).

Data and Data Analysis

Zipf Data Analysis – Page 34 – isef.heers.net/v3/zipf.xlsx

Table 4 – Results of the indexing scheme analysis – Page 46 – isef.heers.net/v3/indexing-scheme-analysis.xlsx

Table 5 – Error Words Analysis



Letter	Occurrence	Part of Text	Adjusted
␑	10520	0.1977	97699
a	4669	0.0877	43361
n	4270	0.0802	39655
r	1806	0.0339	16772
e	1439	0.0270	13364
s	1373	0.0258	12751
f	845	0.0159	7848
i	827	0.0155	7680
o	749	0.0141	6956
m	593	0.0111	5507
g	562	0.0106	5219
t	495	0.0093	4597
h	444	0.0083	4123
l	404	0.0076	3752
c	311	0.0058	2888
p	239	0.0045	2220
d	215	0.0040	1997
y	200	0.0038	1857
b	152	0.0029	1412
u	127	0.0024	1179
w	46	0.0009	427
k	31	0.0006	288
x	13	0.0002	121
z	12	0.0002	111
v	8	0.0002	74
j	0	0.0000	0
q	0	0.0000	0
⍰	0	0.0000	0

To determine where single-letter lexemes belong in the dataset, a series of sentences were tested to find how often a single-letter lexeme is used. Their occurrences were tracked and compiled into Table 4. The test also tracked how many total lexemes were used in the test (53215) and calculated the Part of Text column. Originally, (a) had an occurrence of 43361 in x16web, so using the Part of Text and a multiplier, each letter was mapped to an adjusted occurrence value which is fed into the dataset as specialIDs, to replace the previous placement of single-letter lexemes. Now, (s) is ranked higher, because it is commonly used to build plurals out of singular lexemes. The remnants of this in the code is called errorWords testing, because the lack of some letters caused frequent errors and unencodability.

Sentence Analysis

While previous iterations used arbitrary texts, like the Constitution and The Great Gatsby, to test the performance of the encoder, v3 uses the 300k web sentences dataset from Page 42. It more accurately represents the web15 dataset and provides real-world sentences that this project could be used to encode. Table 6 shows the best, median, and worst 5 sentences from the test.

Table 6 – Shortened Sentence Test Results

Sentence	NLC	ASCII	Ratio
Health and Safety Executive provides further information relating to asbestos.	91	624	6.857
Developed in association with professional membership.	69	432	6.261
What happens following phase 1 consultation?	93	560	6.022
Vale of Glamorgan Council declined to comment.	62	368	5.935
Financial support could be available depending on your circumstances.	93	552	5.935
…			
A bottle of chilled champagne awaits your arrival.	117	400	3.419
Lewis Hertslet was based in West London, moving to Richmond where he lived the majority of his life.	234	800	3.419
The wash basin is produced with two tap holes and is supplied as standard with a combined waste and overflow fitting, plug and chain and fixing studs.	351	1200	3.419
One trader described the Christmas market as ‘absolute rubbish’ with takings well down on last year.	234	800	3.419
This large format stone effect tile is ideal for creating that hotel-feel interior in your own home.	234	800	3.419
…			
It had EVERTHING on it.	175	184	1.051
They were FANTASTIC!	155	160	1.032
Period : from 06:33 on 6/04/2018 to 06:00 on 31/08/2018.	437	448	1.025
HEADLINERS: AXEL BLAKE - sweet boy of comedy!	354	360	1.017
This is NOT a JOB DISGUISED as a BUSINESS!	332	336	1.012

The complete sentence test results are available on isef.heers.net/v3/sentence-testing.xlsx

Table 7 – Sentence Test Summary

Mean Ratio	3.37
Min	1.01
Q1	2.94
Med	3.42
Q3	3.84
Max	6.86

This places v3 significantly higher than v2 in efficiency (v2 would usually only be twice as efficient as ASCII). One can expect NLC to encode text 3.37 times more efficiently than ASCII, with up to 6.86 times on best-case scenarios. The theoretical maximum efficiency of 16.88 times more efficient than ASCII is achieved by encoding the text `"Financial Services Compensation Scheme"`, which has the highest ratio of any singular lexeme in web15. The theoretical worst efficiency, 0.44 times better (2.27 times worse) than ASCII, comes from encoding a single-letter lexeme in the 15-bit index range, such as (±). Realistically, one should expect 2.94 to 3.84 times better than ASCII for any given text encoded.

Sources of Error

This iteration is much more complex than v2 and v1 and has more potential points of failure and undiscovered bugs. Many inconsistencies were discovered during the development process, and there are likely undiscovered issues that can show up in experimentation. Additionally, lexemes that are unencodable are replaced with a SUB (Substitute) character, which can result in a higher ratio, but with lost lexemes. There were no instances of the SUB character in the sentence tests, but a similar bug could potentially cause an irregularity in the data or in custom inputs.

Variations

Other datasets could allow for optimization for specific applications, such as between texting, formal and informal communications, and the distinctive styles of communication between different age groups and demographics, but implementing a new dataset is not well streamlined, and requires very specific procedures that are difficult to implement without knowledge of the intricacies of NLC.

There is a DIF index in preparation for an application of an LLM (Large Language Model) to infer subsequent lexemes. An analogy to this functionality would be the autocomplete suggestion bar on a phone, where if the next lexeme, based on the previous ones, is what the LLM would predict, then the index assignment would be DIF and the ID can be a series of 0s to identify if it is the first, second, third, or *n*th prediction from the LLM. As this would be a significant variation to implement, it will require its own future iteration.

There are some cases where information is lost in the encoding process when the input is not natural language, such as consecutive spaces only being encoded at most as one (will be fixed by detecting and replacing consecutive spaces with DC2 characters); however, there are far more punctuation and characters supported in v3 that would otherwise be an unknown character in v2, such as `"“"`, `"–"`, `"…"`, `"”"`, `"’"`, `"€"`, `"•"`, `"z"`, `"—"`, `"*"`, `">"`, `"»"`, `"`"`, `"<"`, `"·"`, `"™"`, `"«"`, `"‘"`, `"~"`, `"£"`, `"%"`, `"½"`, `"-"`, and `"±"`.

Conclusion

This iteration was very successful and exceeded the highest goal for the project (median 3.42 times more efficient than ASCII). It is functional, provides usable results, and has less semantic loss than v2, as it can encode significantly more punctuation and characters.

There is still much to add to this project; along with the optimizations in Further Research, not all the ideas specified on Page 30 were realized, and many of them can still improve the efficiency of NLC. The demo of this project is available on isef.heers.net/v3 and the subiterations are available on isef.heers.net/v3/subiterations.

End of Development and Iteration Write-Ups

Engineering Paper (continued)

Sources of Error

Iterations 1 and 2 are simpler, consistent, more straightforward approaches, with all known bugs affecting output or statistical calculations addressed. There could remain undiscovered bugs or errors in calculations, but the results and tests remain consistent with the expected outcomes.

The sudden spike at approximately 880 characters in Figure 10 (Page 27) was caused by external factors influencing the available CPU time of the computer used during the test, but the irregularity does not affect the conclusion drawn from the data analysis.

Iteration 3 is much more complex than v2 and v1 and has more potential points of failure and undiscovered bugs. Many inconsistencies were discovered during the development process, and there are likely undiscovered issues that can show up in experimentation. Additionally, lexemes that are unencodable are replaced with a SUB (Substitute) character, which can result in a higher ratio, but with lost lexemes. There were no instances of the SUB character in the sentence tests, but a similar bug could potentially cause an irregularity in the data or in custom inputs.

Further Research

Many changes can be made to improve the efficiency and functionality of the encoder. To address the problem with the SMS test – an important problem as this project could be used for satellite-based personal communication – support for punctuation will need to be adjusted to work for a potential SMS dataset.

Functionality to allow the changing of lexeme datasets to optimize different applications (between texting, official communications, informal communications, and the distinctive styles of communication between different age groups and demographics) would allow for higher versatility and specific application-based fine tuning. Currently, implementing a new dataset is not well streamlined and requires very specific procedures that are difficult to implement without knowledge of the intricacies of NLC.

There is a DIF (111) index in preparation for an application of an LLM (Large Language Model) to infer subsequent lexemes. An analogy to this functionality would be the autocomplete suggestion bar on a phone, where if the next lexeme, based on the previous, is what the LLM would predict, then the index assignment would be DIF and the ID can be a series of 0s to identify if it is the first, second, third, or n^{th} prediction from the LLM. As this would be a significant variation to implement, it will require its own future iteration.

If the former is impractical, the DIF index could instead be used to indicate some special function. An index of DIF could indicate that the next *n* IDs are 5-bit letters, with *n* being defined by the number of 0s following the DIF and terminated by a 1 (including the DIF and termination) (111 0 0 1 could mean the next four IDs are letters). Similarly, an index of DIF could be used to show that the next *n* IDs have the same index, but *n* is the number of 1s and terminated by a 0 (100 010 111 1 1 0 001 instead of 100 010 010 010 010 010 001).

In v3, some information is lost in the encoding process when the input is not natural language, such as consecutive spaces only being encoded at most as one, and this can be addressed in future iterations.

Conclusion

The goal was to create an encoder for NL, more efficient than ASCII, and that goal was achieved. Already in the first iteration, NLC was consistently over 1.5x as efficient, even on the worst-case texts. However, 1.5x efficiency was only the minimum goal, and the highest goal aimed for 3x efficiency.

Efficiency-wise there was only a minor improvement in v2 over v1, but almost every semantic loss problem with v1 was addressed, except for undefined character encoding. Undefined words became whole and not atomized, uppercase characters were encoded uppercased, spaces could be in any quantity, and newlines and spaces were rendered according to the original input.

V2 did not reach much higher compression ratios than v1, and it was on average only twice as efficient as ASCII, however, this is because v1 encoded less information than v2 did. V2 could be considered a success as it was only slightly better than v1 in efficiency, but had large gains on encoded information and convenience of use over v1.

Iteration 3 was successful and exceeded the highest goal for the project (median 3.42 times more efficient than ASCII). It is functional, provides usable results, and has less semantic loss than v2, as it can encode significantly more punctuation and characters.

There is still much to add to this project; along with the optimizations in Further Research, not all the ideas specified on Page 30 were realized, and many of them can still improve the efficiency of NLC.

Bibliography

Chen, T., & Kan, M.-Y. (2013). Creating a Live, Public Short Message Service Corpus: The NUS SMS Corpus. *Language Resources and Evaluation*, *47*(2), 299–355. https://doi.org/10.1007/s10579-012-9197-9

Corpus used for testing SMS messages on Iterations 1 and 2.

Khalid Sayood. (2000). *Introduction to data compression* (2nd ed.). Morgan Kaufmann Publishers.

Book used to learn about fundamental data compression concepts, basic information theory, and some terminology regarding data encoding.

Mahmood, Md. A., & Hasan, K. M. A. (2019, November 1). *A Dictionary based Compression Scheme for Natural Language Text with Reduced Bit Encoding*. IEEE Xplore; IEEE. https://doi.org/10.1109/RAAICON48939.2019.62

A similar paper that uses a character-based encoding approach using dictionary compression. It uses an encoding scheme called 5-Bit Compression, which can encode characters from various languages in 5 bits. NLC uses lexeme-based, instead of character-based encoding, and with a different approach to dictionary compression.

Mahmood, Md. A., & Hasan, K. M. A. (2022). An Efficient Compression Scheme for Natural Language Text by Hashing. *SN Computer Science*, *3*(4). https://doi.org/10.1007/s42979-022-01210-0

Compresses text by encoding common sequences of letters as single units. Similar to what NLC must do when a word needs to be split into lexemes.

Piantadosi, S. T. (2014). Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic Bulletin & Review*, *21*(5). https://doi.org/10.3758/s13423-014-0585-6

Used to learn about specific features of Zipf's law, terminology, and better understanding to aid in implementation.

Tatman, R. (2017). *English Word Frequency*. Kaggle. https://www.kaggle.com/datasets/rtatman/english-word-frequency/data

Corpus used for the web333k model.

Universität Leipzig. (2023). *Download Corpora English*. Wortschatz Leipzig; Deutscher Wortschatz / Wortschatz Leipzig. https://wortschatz.uni-leipzig.de/en/download/english

Various corpora used for Iteration 3.

Wikipedia Contributors. (2016). *Frequency lists/English/Wikipedia (2016)*. Wiktionary. https://en.wiktionary.org/wiki/Wiktionary:Frequency_lists/English/Wikipedia_(2016)

Corpus used for the wiki1k model.

Iterations' Source Code

Iteration 1

./index.html

```html
<!DOCTYPE html>
<html>

<head>
  <script src="./p5.js"></script>
  <script src="./wordfreq-333k.js"></script>
  <title>Natural Language Encoder</title>
  <link rel="stylesheet" href="darkmode.css">
  <style>
    .input-container {
      padding-right: 7px;
    }
    textarea {
      width: 100%;
      height: 100px;
      resize: vertical;
      font-size: medium;
    }
    p {
      font-family: monospace;
      font-size: medium;
      word-wrap: break-word;
    }
  </style>
</head>

<body>
  <main>
    <h1>Natural Language Encoder v1</h1>
    <p id="stats"></p><br>
    <div class="input-container">
      <label for="og">Natural Language Input</label>
      <textarea id="og" name="og" onKeyUp="refresh()"></textarea>
    </div>
    <div class="output-container">
      <label for="outin">Original</label>
      <p type="text" id="outin" name="outin"></p>
      <label for="out">Interpretation</label>
      <p type="text" id="out" name="out"></p>
      <label for="bitarray">NLC Encoded</label>
      <p type="text" id="bitarray" name="bitarray"></p>
      <label for="bitarray">ASCII Encoded</label>
      <p type="text" id="ascii" name="ascii"></p>
    </div>
    <script src="./script.js"></script>
  </main>
</body>
</html>
```

./script.js

```
const specialIDs = [ // Letters sorted by frequency on
https://en.wikipedia.org/wiki/Letter_frequency
    "e",
    "t",
    "a",
    "o",
    "n",
    "r",
    "i",
    "s",
    "h",
    "d",
    "l",
    "f",
    "c",
    "m",
    "u",
    "g",
    "y",
    "p",
    "w",
    "b",
    "v",
    "k",
    "j",
    "x",
    "z",
    "q",
    "0",
    "1",
    "2",
    "3",
    "4",
    "5",
    "6",
    "7",
    "8",
    "9",
    "�",
    ".",
    ",",
    "-",
    "'",
    "\"",
    "$",
    "(",
    ")",
    ":",
    "?",
    "/",
    "!",
    "&",
    ";",
    "\n"
]
const punctuation = [
    ".",
    ",",
    "-",
    "'",
    "\"",
    "$",
    "(",
    ")",
    ":",
    "?",
    "/",
    "!",
    "&",
    ";",
    "\n"
]

let encoded = ""
let decoded = []
let formattedarray = []

const wordsbitsize = Math.ceil(dec2bin(words.length, 0).length / 4)

function setup() {
    refresh()
}

function draw() {
    //    refresh()
}

function refresh() {
    let og = ""
    let ogarray = []
    let idArray = []
    formattedarray = []
    let bitArrayIndex = []
    let bitArray = []
    let idbitsize = 0

    og = document.getElementById("og").value
    ogarray = og.split(" ")
    for (let i = 0; i < ogarray.length; i++) {
        /*
        You => you => 310
        asDf => asdf => 2,7,9,11
        */
        if (words.indexOf(ogarray[i].toLowerCase()) == -1) {
            let wordarray = ogarray[i].split('')
            for (let j = 0; j < wordarray.length; j++) {
                if (specialIDs.indexOf(wordarray[j].toLowerCase()) == -1) {
                    idArray.push(specialIDs.indexOf("�"))
                } else {
                    idArray.push(specialIDs.indexOf(wordarray[j].toLowerCase()))
                }
            }
            formattedarray.push(`<span style="color: red;">${ogarray[i]}</span>`)
        } else {
            idArray.push(words.indexOf(ogarray[i].toLowerCase()) + specialIDs.length)
            idbitsize = Math.ceil(dec2bin(words.indexOf(ogarray[i].toLowerCase()) +
specialIDs.length, 0).length / 4)
            formattedarray.push(`<span style="color: rgb(${(idbitsize) * 255 /
wordsbitsize},255,${(idbitsize) * 255 / wordsbitsize}">${ogarray[i]}</span>`)
        }
    }

    for (let i = 0; i < idArray.length; i++) {
        bitArrayIndex.push(dec2bin(Math.ceil(dec2bin(idArray[i], 0).length / 4), 3))
    }
    bitArrayIndex.push("000")
    for (let i = 0; i < idArray.length; i++) {
        /*
        dec2bin(310,0) =>
        "100110110" length = 9 (lead to the nearest 4-bit boundary: should be 12)
        4*floor(x/4+1)
        9 => 12
        dec2bin(310,12) => "000100110110" length = 9
        */
        bitArray.push(dec2bin(idArray[i], 4 * Math.ceil(dec2bin(idArray[i], 0).length / 4)))
    }

    encoded = bitArrayIndex.join("") + bitArray.join("")
    decode()
    document.getElementById("bitarray").innerHTML = encoded
    document.getElementById("outin").innerHTML = formattedarray.join(" ")
    document.getElementById("out").innerHTML = decoded.join(" ")
    document.getElementById("ascii").innerHTML = ascii2bin(og)

    document.getElementById("stats").innerHTML =
        "<br>NLC Size - " + encoded.length +
        "<br>ASCII Size - " + ascii2bin(og).length +
        "<br>Ratio - " + ascii2bin(og).length / encoded.length + " : 1"


}

function decode() {
    let bitArrayIndex = []
    let bitArray = []
    let i = 0
    decoded = []
    while (true) {
        if (encoded.substring(3 * i, 3 * i + 3) == "000") break
        bitArrayIndex.push(bin2dec(encoded.substring(3 * i, 3 * i + 3)))
        i++
    }
    let startpoint = bitArrayIndex.length * 3 + 3
    for (let i = 0; i < bitArrayIndex.length; i++) {
        bitArray.push(encoded.substring(startpoint, startpoint + 4 * bitArrayIndex[i]))
        startpoint = startpoint + 4 * bitArrayIndex[i]
    }
    for (let i = 0; i < bitArray.length; i++) {
        decoded.push(words[bin2dec(bitArray[i]) - specialIDs.length])
        decoded.push(specialIDs[bin2dec(bitArray[i])])
    }
}

function dec2bin(dec, length) {
    // returns binary of decimal. dec is decimal input. length determines necessary leading
zeroes
    // ex. dec2bin(19, 8) => 00010011 dec2bin(19,0) => 10011
    let bin = ""
    while (true) {
        bin = dec % 2 + bin
        dec = Math.floor(dec / 2)
        if (dec == 0) break
    }
    let binlen = bin.length
    for (let i = 0; i < length - binlen; i++) {
        bin = 0 + bin
    }
    return (bin)
}

function bin2dec(bin) {
    let binarray = bin.split("")
    let dec = 0
    for (let i = 0; i < binarray.length; i++) {
        dec += Math.pow(2, i) * binarray[binarray.length - i - 1]
    }
    return (dec)
}

function ascii2bin(input) {
    // Credit: https://www.geeksforgeeks.org/javascript-program-to-convert-a-string-to-binary/
    let binaryResult = "";
    for (let i = 0; i < input.length; i++) {
        const charCode = input.charCodeAt(i);
        let binaryValue = "";
        for (let j = 7; j >= 0; j--) {
            binaryValue += (charCode >> j) & 1;
        }
        binaryResult += binaryValue;
    }
    return binaryResult.trim();
}
```

./darkmode.css

```
.jfk-bubble.gtx-bubble,

.captcheck_answer_label>input+img,
span#closed_text>img[src^="https://www.gstatic.com/images/branding/googlelogo"],
span[data-href^="https://www.hcaptcha.com/"]>#icon,
img.Wirisformula {
    filter: invert(100%) hue-rotate(180deg) contrast(90%) !important;
}

[data-darkreader-inline-bgcolor] {
    background-color: var(--darkreader-inline-bgcolor) !important;
}

[data-darkreader-inline-bgimage] {
    background-image: var(--darkreader-inline-bgimage) !important;
}

[data-darkreader-inline-border] {
    border-color: var(--darkreader-inline-border) !important;
}

[data-darkreader-inline-border-bottom] {
    border-bottom-color: var(--darkreader-inline-border-bottom) !important;
}

[data-darkreader-inline-border-left] {
    border-left-color: var(--darkreader-inline-border-left) !important;
}

[data-darkreader-inline-border-right] {
    border-right-color: var(--darkreader-inline-border-right) !important;
}

[data-darkreader-inline-border-top] {
    border-top-color: var(--darkreader-inline-border-top) !important;
}

[data-darkreader-inline-boxshadow] {
    box-shadow: var(--darkreader-inline-boxshadow) !important;
}

[data-darkreader-inline-color] {
    color: var(--darkreader-inline-color) !important;
}

[data-darkreader-inline-fill] {
    fill: var(--darkreader-inline-fill) !important;
}

[data-darkreader-inline-stroke] {
    stroke: var(--darkreader-inline-stroke) !important;
}

[data-darkreader-inline-outline] {
    outline-color: var(--darkreader-inline-outline) !important;
}

[data-darkreader-inline-stopcolor] {
    stop-color: var(--darkreader-inline-stopcolor) !important;
}

[data-darkreader-inline-bg] {
    background: var(--darkreader-inline-bg) !important;
}

[data-darkreader-inline-invert] {
    filter: invert(100%) hue-rotate(180deg);
}

:root {
    --darkreader-neutral-background: #181a1b;
    --darkreader-neutral-text: #e8e6e3;
    --darkreader-selection-background: #004daa;
    --darkreader-selection-text: #e8e6e3;
}

@layer {
    html {
        background-color: #181a1b !important;
    }

    html {
        color-scheme: dark !important;
    }

    iframe {
        color-scheme: dark !important;
    }

    html,
    body {
        background-color: #181a1b;
    }

    html,
    body {
        border-color: #736b5e;
        color: #e8e6e3;
    }

    a {
        color: #3391ff;
    }

    table {
        border-color: #545b5e;
    }

    mark {
        color: #e8e6e3;
    }

    ::placeholder {
        color: #b2aba1;
    }

    input:-webkit-autofill,
    textarea:-webkit-autofill,
    select:-webkit-autofill {
        background-color: #404400 !important;
        color: #e8e6e3 !important;
    }

    * {
        scrollbar-color: #454a4d #202324;
    }

    ::selection {
        background-color: #004daa !important;
        color: #e8e6e3 !important;
    }

    ::-moz-selection {
        background-color: #004daa !important;
        color: #e8e6e3 !important;
    }
}

textarea {
    width: 100%;
    height: 100px;
    /* Adjust height as needed */
    resize: vertical;
    /* Allow vertical resizing */
    font-size: medium;
}

p {
    font-family: monospace;
    font-size: medium;
    word-wrap: break-word;
}

.vimvixen-hint {
    background-color: #684b00 !important;
    border-color: #9e7e00 !important;
    color: #d7d4cf !important;
}

#vimvixen-console-frame {
    color-scheme: light !important;
}

::placeholder {
    opacity: 0.5 !important;
}

#edge-translate-panel-body,
.MuiTypography-body1,
.nfe-quote-text {
    color: var(--darkreader-neutral-text) !important;
}

gr-main-header {
    background-color: #1b4958 !important;
}

.tou-z65h9k,
.tou-mignzq,
.tou-1b6i2ox,
.tou-lnqlqk {
    background-color: var(--darkreader-neutral-background) !important;
}

.tou-75mvi {
    background-color: #0f3a47 !important;
}

.tou-ta9e87,
.tou-1w3fhi0,
.tou-1b8t2us,
.tou-py7lfi,
.tou-1lpmd9d,
.tou-1frrtv8,
.tou-17ezmgn {
    background-color: #1e2021 !important;
}

.tou-uknfeu {
    background-color: #432c09 !important;
}

.tou-6i3zyv {
    background-color: #245d70 !important;
}

div.mermaid-viewer-control-panel .btn {
    background-color: var(--darkreader-neutral-background);
    fill: var(--darkreader-neutral-text);
}

svg g rect.er {
    fill: var(--darkreader-neutral-background) !important;
}

svg g rect.er.entityBox {
    fill: var(--darkreader-neutral-background) !important;
}

svg g rect.er.attributeBoxOdd {
    fill: var(--darkreader-neutral-background) !important;
}

svg g rect.er.attributeBoxEven {
    fill: var(--darkreader-selection-background);
    fill-opacity: 0.8 !important;
}

svg rect.er.relationshipLabelBox {
    fill: var(--darkreader-neutral-background) !important;
}

svg g g.nodes rect,
svg g g.nodes polygon {
    fill: var(--darkreader-neutral-background) !important;
}

svg g rect.task {
    fill: var(--darkreader-selection-background) !important;
}
```

```
svg line.messageLine0,
svg line.messageLine1 {
    stroke: var(--darkreader-neutral-text) !important;
}

div.mermaid .actor {
    fill: var(--darkreader-neutral-background) !important;
}

mitid-authenticators-code-app>.code-app-container {
    background-color: white !important;
    padding-top: 1rem;
}

iframe#unpaywall[src$="unpaywall.html"] {
    color-scheme: light !important;
}

embed[type="application/pdf"] {
    filter: invert(100%) contrast(90%);
}
```

Iteration 2

./index.html

```
<!DOCTYPE html>
<html>

<head>
  <script src="./p5.js"></script>
  <script src="./1k-wikipedia.js"></script>
  <script src="./333k-web.js"></script>
  <script src="./eng_news_2023_10K-sentences.js"></script>
  <title>Natural Language Encoder</title>
  <link rel="stylesheet" href="darkmode.css">
  <style>
    .input-container {
      padding-right: 7px;
    }
    textarea {
      width: 100%;
      height: 100px;
      resize: vertical;
      font-size: medium;
    }
    p {
      font-family: monospace;
      font-size: medium;
      word-wrap: break-word;
      white-space: pre-wrap;
    }
    .dataAnalysis {
      display: none;
    }
    h2 {
      font-size: 1.25em;
      margin-bottom: 0;
    }
  </style>
</head>
<body>
  <main>
    <a href="./decode">NLC Decoder</a><br>
    <a href="./progressive-compute/">Progressive-Compute</a>
    <h1>Natural Language Encoder v2</h1>
    <p id="stats"></p>
    <div class="dropdown-container">
      <label for="wordModel-select">NLC Word Model:</label>
      <select id="wordModel-select" name="wordModel-select" onchange="changeWordModel()">
        <option value="web333k">333k-web</option>
        <option value="wikipedia1k">1k-wikipedia</option>
      </select>
    </div><br>
    <div class="input-container">
      <label for="og">Natural Language Input</label>
      <textarea id="og" name="og" onKeyUp="refresh()"></textarea>
    </div>
    <div class="output-container">
      <h2 for="outin">Original</h2>
      <label for="toggle-brackets">Render Brackets</label>
      <input type="checkbox" id="toggle-brackets" name="toggle-brackets"
onchange="toggleBrackets()"><br>
```

```
      <label for="toggle-xon">Render Uppercase Control Character</label>
      <input type="checkbox" id="toggle-xon" name="toggle-xon" onchange="toggleXON()"
checked="true">
      <p type="text" id="outin" name="outin"></p>
      <h2 for="out">Interpretation</h2>
      <p type="text" id="out" name="out"></p>
      <h2 for="bitarray">
        NLC Encoded
        <button onclick="copyBitArray()">Copy</button>
      </h2>
      <p type="text" id="bitarray" name="bitarray"></p>
      <h2 for="bitarray">ASCII Encoded</h2>
      <p type="text" id="ascii" name="ascii"></p>
    </div>
    <label for="dataAnalysis" class="dataAnalysis">For data analysis</label>
    <div id="dataAnalysis" class="dataAnalysis">
      <button onclick="copyWords()">Copy</button>
      <button onclick="startTimeTest()">Start time measurement</button>
    </div>
    <script src="./script.js"></script>
  </main>
</body>
</html>
```

./script.js

```
// Letters sorted by frequency on https://en.wikipedia.org/wiki/Letter_frequency
const specialIDs = [ "e", "t", "a", "o", "n", "r", "i", "s", "h", "d", "l", "f", "c", "m",
"u", "g", "y", "p", "w", "b", "v", "k", "j", "x", "z", "q", "0", "1", "2", "3", "4", "5", "6",
"7", "8", "9", "�" ]
const XON = "␑" // Uppercase control (ASCII Device Control One (XON)) is set as a constant,
because ␑ is difficult to type and not monospaced
const punctuation = [
    ".",
    ",",
    "-",
    "'",
    "\"",
    "$",
    "(",
    ")",
    ":",
    "?",
    "/",
    "!",
    ";",
    "\n",
    " ",
    XON,
]

let encoded = ""
let decoded = []
let formattedArray = []
let formattedBitArray = []

const maxRatio = 9 // Aesthetics (Calculated manually dependent on dataset)
const blueThreshold = 3 // Aesthetics

let brackets = false
let renderXON = true

let startTime
let endTime
let timeDelta

/*
let timeTesting = false
let iteration = 0
let timeData = [[0]]
*/

let words = []

let wordModelGlobal = ""

function setup() {
    changeWordModel()
    toggleBrackets()
    refresh()
}

function changeWordModel() {
    wordModelGlobal = document.getElementById("wordModel-select").value
    words = eval(wordModelGlobal)
    refresh()
}

function toggleBrackets() {
    brackets = document.getElementById("toggle-brackets").checked
    refresh()
}

function copyBitArray() {
    if (brackets) navigator.clipboard.writeText(formattedBitArray.join(""));
    if (!brackets) navigator.clipboard.writeText(encoded);
}

function toggleXON() {
    renderXON = document.getElementById("toggle-xon").checked
    refresh()
}

function draw() {
    //startTimeTest()
}

function copyWords() {
    //let wordRatios = []
    //for (let i = 0; i < 10; i++) {
    //  wordRatios.push([words[i], words[i].length * 2, Math.ceil(dec2bin(i +
punctuation.length + specialIDs.length, 0).length / 4)].join("\t"))
    //}
    //navigator.clipboard.writeText(wordRatios.join("\n"));

    navigator.clipboard.writeText(timeData.join("\n").replaceAll(",", "\t"));
}

function startTimeTest() {
    timeTesting = true
    console.log(timeData);
    document.getElementById("og").value = news.substring(0,iteration)
    refresh();
    timeData[iteration].push(timeDelta)
    if(timeData[iteration].length == 5) {
        iteration+= 1
        timeData.push([iteration])
    }
}

function refresh() {
    startTime = Date.now()
    reset = false
    let wordModel = wordModelGlobal
    let og = ""
    let idArray = []
    formattedArray = []
    formattedBitArray = []
    let bitArray = []

    let simplicity = false
    let punctuationOrder = 0

    og = document.getElementById("og").value
    let raw = og.replaceAll(/([A-Z])/g, "␑$1").toLowerCase() // Replace uppercase characters
with upppercase control characters
    let rawArray = []
    let sp = 0
    let i = 0
    let punct = ""

    // Convert og to rawArray (seperate by space and punctuation)
    while (i < raw.length) {
        while (!(punctuation.includes(raw.charAt(i)))) { // Increase i until a punctuation is
reached
            i++
            if (i >= raw.length) break // Protects against freezing
        }
        punct = raw.charAt(i)
        if (punct == " ") {
            if (punctuation.includes(raw.charAt(i + 1))) {
                rawArray.push(raw.substring(sp, i), " ") // Spaces are not ignored if a
punctuation is after the space (He said, "Hello World.")
            } else {
                rawArray.push(raw.substring(sp, i)) // Spaces are ignored
            }
        } else {
            rawArray.push(raw.substring(sp, i), punct) // Punctuation with no preceding spaces
are added
        }
        i++
        while (punctuation.includes(raw.charAt(i))) { // Accounts for all punctuation after
1st punctuation
            punct = raw.charAt(i)
            rawArray.push(punct)
            i++
        }
        sp = i
    }

    // Assign identifiers
    for (let i = 0; i < rawArray.length; i++) {
        if (punctuation.includes(rawArray[i])) {
            // Check punctuation
            idArray.push(punctuation.indexOf(rawArray[i]))
            if (brackets) formattedArray.push(`<span style="color:
#ffd700;">[${rawArray[i].replace("\n", "<br>")}]</span>`)
            if (!brackets) formattedArray.push(`<span style="color:
#ffd700;">${rawArray[i].replace("\n", "<br>").replace(XON, renderXON ? XON : "")}</span>`)
        } else if (words.includes(rawArray[i])) {
            // Check words
            idArray.push(words.indexOf(rawArray[i]) + punctuation.length + specialIDs.length)

            // Format text
            let idNybbleSize = Math.ceil(dec2bin(words.indexOf(rawArray[i].toLowerCase()) +
punctuation.length, 0).length / 4) + 0.75
            let asciiNybbleSize = ascii2bin(rawArray[i]).length / 4
            let x = asciiNybbleSize / idNybbleSize // Ratio
            let t = blueThreshold
            let m = maxRatio
            if (brackets) formattedArray.push(`<span style="color: rgb(${x < t ? 255 - 255 * x
/ t : 0}, 255, ${x >= t ? 255 * (x - t) / (m - t) : 255 - 255 * x /
t});">(${rawArray[i]})</span>${!(punctuation.includes(rawArray[i + 1])) && rawArray[i + 1] !=
"" ? " " : ""}`) // Adds text to formattedArray, along with whitespace if there is not
following puntuation
```

```
            if (!brackets) formattedArray.push(`<span style="color: rgb(${x < t ? 255 - 255 *
x / t : 0}, 255, ${x >= t ? 255 * (x - t) / (m - t) : 255 - 255 * x /
t});">${rawArray[i]}</span>${!(punctuation.includes(rawArray[i + 1])) && rawArray[i + 1] != ""
? " " : ""}`)
            /* Color control for words:
                Channels
                    Red
                        Linearly decreases from (0,255) to (blueThreshold, 0)
                        Remains constant at 0 after blueThreshold
                    Green
                        Remains constant at 255
                    Blue
                        Follows red channel until blueThreshold
                        Linearly increases from (blueThreshold, 0) to (maxRatio, 255)
                Effect
                    Low ratios are whiter, and become greener as the ratio increases until the
blueThreshold. After that, they become cyaner until maxRatio, where it stays at pure rgb(0,
255, 255).
                https://www.desmos.com/calculator/lz5uv3al8y
            */
        } else if (rawArray[i] != "") {
            // Create words from letters, numbers, and unknown character
            for (let j = 0; j < rawArray[i].length; j++) {
                idArray.push(specialIDs.indexOf(rawArray[i][j]) + punctuation.length)
                if(!specialIDs.includes(rawArray[i][j])) idArray.push(specialIDs.indexOf("�")
+ punctuation.length)
            }
            if (brackets) formattedArray.push(`<span style="color: rgb(255, 0,
0);">{${rawArray[i]}}</span>`)
            if (!brackets) formattedArray.push(`<span style="color: rgb(255, 0,
0);">${rawArray[i]}</span>`)
            if (!punctuation.includes(rawArray[i + 1])) {
                idArray.push(punctuation.indexOf(" "))
                if (!brackets) formattedArray.push(" ")
                if (brackets) formattedArray.push('<span style="color: #ffd700;">[ ]</span>')
            }
        }
    }
"Hello"

    // Create binary datastream

    // Control (could be useful in future iterations, disabled for now)
    //bitArray.push(
    //  simplicity ? 1 : 0,
    //  wordModel == "web333k" ? 0 : 1,
    //  punctuationOrder
    //)
    //if (brackets) formattedBitArray.push(simplicity ? 1 : 0, wordModel == "web333k" ? 0 : 1,
punctuationOrder, " - ")

    // Indexes
    for (let i = 0; i < idArray.length; i++) {
        bitArray.push(dec2bin(Math.ceil(dec2bin(idArray[i], 0).length / 4), 3))
        if (brackets) formattedBitArray.push(dec2bin(Math.ceil(dec2bin(idArray[i], 0).length /
4), 3), " ")
    }

    bitArray.push("000")
    if (brackets) formattedBitArray.push("(000) - ")
    for (let i = 0; i < idArray.length; i++) {
        /*
        dec2bin(310,0) =>
        "100110110" length = 9 (lead to the nearest 4-bit boundary: should be 12)
        4*floor(x/4+1)
        9 => 12
        dec2bin(310,12) => "000100110110" length = 9
        */
        bitArray.push(dec2bin(idArray[i], 4 * Math.ceil(dec2bin(idArray[i], 0).length / 4)))
        if (brackets) {
            for (let j = 0; j < bitArray[bitArray.length - 1].length / 4; j++) {
                formattedBitArray.push(bitArray[bitArray.length - 1].substring(j * 4, j * 4 +
4), " ")
            }
            if (i + 1 < idArray.length) formattedBitArray.push(" | ")
        }
    }

    encoded = bitArray.join("")
    decode()
    if (brackets) document.getElementById("bitarray").innerHTML = formattedBitArray.join("")
    if (!brackets) document.getElementById("bitarray").innerHTML = encoded
    document.getElementById("outin").innerHTML = formattedArray.join("")
    document.getElementById("out").innerHTML = decoded
    document.getElementById("ascii").innerHTML = ascii2bin(og)

    endTime = Date.now()
    timeDelta = endTime-startTime

    document.getElementById("stats").innerHTML =
        "<br>NLC Size - " + encoded.length +
        "<br>ASCII Size - " + ascii2bin(og).length +
        "<br>Ratio - " + ascii2bin(og).length / encoded.length + " : 1" +
        "<br>Time - " + timeDelta + "ms"

}

function isUppercase(char) {
    return char.toUpperCase() == char
}

function decode() {
    let indexArray = []
    let idArray = []
    let idLookup = [...punctuation, ...specialIDs, ...words]
    decoded = []

    // Control
    //let simplicity = encoded[0] == 0 ? false : true
    //if (!simplicity) {
    //  let wordModel = encoded[1] == 0 ? "web333k" : "wikipedia1k"
    //  let punctuationOrder = encoded[2]
    //}

    // Indexes
    for (let i = 0; true; i++) {
        if (encoded.substring(3 * i, 3 * i + 3) == "000") break
        indexArray.push(bin2dec(encoded.substring(3 * i, 3 * i + 3)))
    }

    // IDs
    let j = indexArray.length * 3
    for (let i = 0; i < indexArray.length; i++) {
        idArray.push(bin2dec(encoded.substring(j + 3, j + 4 * indexArray[i] + 3)))
        j = j + 4 * indexArray[i]
    }

    // Convert IDs to content
    for (let i = 0; i < idArray.length; i++) {
        decoded.push(idLookup[idArray[i]].replace("\n", "<br>"))

        // If word followed by word, place space
        if (words.includes(idLookup[idArray[i]]) && (words.includes(idLookup[idArray[i + 1]])
|| specialIDs.includes(idLookup[idArray[i+1]]))) decoded.push(" ")
    }

    // Create uppercases
    decoded = decoded.join("").split("") // Turn into an array split by characters
    for(let i = 0; i < decoded.length; i++) {
        if (decoded[i] == XON) {
            decoded.splice(i,1)
            decoded[i] = decoded[i].toUpperCase()
        }
    }
    decoded = decoded.join("")
}

function dec2bin(dec, length) {
    // returns binary of decimal. dec is decimal input. length determines necessary leading
zeroes
    // ex. dec2bin(19, 8) => 00010011 dec2bin(19,0) => 10011
    let bin = ""
    while (true) {
        bin = dec % 2 + bin
        dec = Math.floor(dec / 2)
        if (dec == 0) break
    }
    let binlen = bin.length
    for (let i = 0; i < length - binlen; i++) {
        bin = 0 + bin
    }
    return (bin)
}

function bin2dec(bin) {
    let binarray = bin.split("")
    let dec = 0
    for (let i = 0; i < binarray.length; i++) {
        dec += Math.pow(2, i) * binarray[binarray.length - i - 1]
    }
    return (dec)
}


function ascii2bin(input) {
    // Credit: https://www.geeksforgeeks.org/javascript-program-to-convert-a-string-to-binary/
    let binaryResult = "";

    for (let i = 0; i < input.length; i++) {
        const charCode = input.charCodeAt(i);
        let binaryValue = "";

        for (let j = 7; j >= 0; j--) {
            binaryValue += (charCode >> j) & 1;
        }

        binaryResult += binaryValue;
    }

    return binaryResult.trim();
}
```

./decode/index.html

```html
<!DOCTYPE html>
<html>

<head>
  <script src="../p5.js"></script>
  <script src="../1k-wikipedia.js"></script>
  <script src="../333k-web.js"></script>
  <title>Natural Language Decoder</title>
  <link rel="stylesheet" href="../darkmode.css">
  <style>
    .input-container {
      padding-right: 7px;
    }
    textarea {
      width: 100%;
      height: 100px;
      resize: vertical;
      font-size: medium;
    }
    p {
      font-family: monospace;
      font-size: medium;
      word-wrap: break-word;
    }
    .dataAnalysis {
      display: none;
    }

    h2 {
      font-size: 1.25em;
      margin-bottom: 0;
    }
  </style>
</head>

<body>
  <main>
    <a href="./..">Natural Language Encoder</a>
    <h1>NLC Decoder v2</h1>
    <p id="stats"></p>
    <div class="dropdown-container">
      <label for="wordModel-select">NLC Word Model:</label>
      <select id="wordModel-select" name="wordModel-select" onchange="changeWordModel()">
        <option value="web333k">333k-web</option>
        <option value="wikipedia1k">1k-wikipedia</option>
      </select>
    </div><br>
    <div class="input-container">
      <label for="og">NLC Binary Input</label>
      <textarea id="og" name="og" onKeyUp="refresh()">0110100001001100011101010100100</textarea>
    </div>
    <div class="output-container">
      <h2 for="outin">Output</h2>
      <p type="text" id="outin" name="outin"></p>
    </div>
    <script src="./script.js"></script>
  </main>
</body>
</html>
```

./decode/script.js

```
// Letters sorted by frequency on https://en.wikipedia.org/wiki/Letter_frequency
const specialIDs = [ "e", "t", "a", "o", "n", "r", "i", "s", "h", "d", "l", "f", "c", "m",
"u", "g", "y", "p", "w", "b", "v", "k", "j", "x", "z", "q", "0", "1", "2", "3", "4", "5", "6",
"7", "8", "9", "�" ]
const XON = "␑" // Uppercase control (ASCII Device Control One (XON)) is set as a constant,
because ␑ is difficult to type and not monospaced
const punctuation = [
    ".",
    ",",
    "-",
    "'",
    "\"",
    "$",
    "(",
    ")",
    ":",
    "?",
    "/",
    "!",
    ";",
    "\n",
    " ",
    XON,
]
let encoded = ""
let decoded = []

let words = []

let wordModelGlobal = ""

function setup() {
    changeWordModel()
    refresh()
}

function changeWordModel() {
    wordModelGlobal = document.getElementById("wordModel-select").value
    words = eval(wordModelGlobal)
    refresh()
}

function refresh() {
    let encoded = document.getElementById("og").value
    let indexArray = []
    let idArray = []
    let idLookup = [...punctuation,...specialIDs,...words]
    decoded = []

    // Control
    //let simplicity = encoded[0] == 0 ? false : true
    //if (!simplicity) {
    //  let wordModel = encoded[1] == 0 ? "web333k" : "wikipedia1k"
    //  let punctuationOrder = encoded[2]
    //}

    // Indexes

    for (let i = 0; true; i++) {
        if (encoded.substring(3 * i, 3 * i + 3) == "000") break
        indexArray.push(bin2dec(encoded.substring(3 * i, 3 * i + 3)))
        if (i == encoded.length) {
            document.getElementById("outin").innerHTML = "Parsing error (No stop code)"
            return // Prevent freezing if binary is invalid
        }
    }

    // IDs
    let j = indexArray.length * 3 + 3
    for (let i = 0; i < indexArray.length; i++) {
        idArray.push(bin2dec(encoded.substring(j, j + 4 * indexArray[i])))
        j = j + 4 * indexArray[i]
    }

    // Convert IDs to content
    for (let i = 0; i < idArray.length; i++) {

        decoded.push(idLookup[idArray[i]].replace("\n", "<br>"))

        // If word followed by word, place space
        if (words.includes(idLookup[idArray[i]]) && (words.includes(idLookup[idArray[i + 1]])
|| specialIDs.includes(idLookup[idArray[i+1]]))) decoded.push(" ")
    }

// Create uppercases
decoded = decoded.join("").split("") // Turn into an array split by characters
for(let i = 0; i < decoded.length; i++) {
    if (decoded[i] == XON) {
        decoded.splice(i,1)
        decoded[i] = decoded[i].toUpperCase()
    }
}
decoded = decoded.join("")

document.getElementById("outin").innerHTML = decoded
}

function bin2dec(bin) {
    let binarray = bin.split("")
    let dec = 0
    for (let i = 0; i < binarray.length; i++) {
        dec += Math.pow(2, i) * binarray[binarray.length - i - 1]
    }
    return (dec)
}
```

./progressive-compute/index.html

```
<!DOCTYPE html>
<html>
<head>
  <script src="../p5.js"></script>
  <script src="../1k-wikipedia.js"></script>
  <script src="../333k-web.js"></script>
  <title>Natural Language Compression</title>
  <link rel="stylesheet" href="../darkmode.css">
  <style>
    .input-container {
      padding-right: 7px;
    }
    textarea {
      width: 100%;
      height: 100px;
      resize: vertical;
      font-size: medium;
    }
    p {
      font-family: monospace;
      font-size: medium;
      word-wrap: break-word;
    }
    .dataAnalysis {
      display: none;
    }
    h2 {
      font-size: 1.25em;
      margin-bottom: 0;
    }
  </style>
</head>
<body>
  <main>
    <a href="../">Natural Language Compression</a>
    <h1>Natural Language Compression v2.0 (Progressive-Compute)</h1>
    <p id="stats"></p>
    <div class="dropdown-container">
      <label for="wordModel-select">NLC Word Model:</label>
      <select id="wordModel-select" name="wordModel-select" onchange="changeWordModel()">
        <option value="web333k">333k-web</option>
        <option value="wikipedia1k">1k-wikipedia</option>
      </select>
    </div><br>
    <div class="input-container">
      <label for="og">Natural Language Input</label>
      <button onclick="startCompute()">Compute</button>
      <textarea id="og" name="og" onKeyUp="refresh()">Note: Progressive-Compute divides
processing between frames, and should only be used on under-powered systems if the main
encoder crashes. This is much slower</textarea>
    </div>
    <div class="output-container">
      <h2 for="outin">Original</h2>
      <label for="toggle-brackets">Render Brackets</label>
      <input type="checkbox" id="toggle-brackets" name="toggle-brackets"
onchange="toggleBrackets()"><br>
      <label for="toggle-xon">Render Uppercase Control Character</label>
      <input type="checkbox" id="toggle-xon" name="toggle-xon" onchange="toggleXON()"
checked="true">
```

```
      <p type="text" id="outin" name="outin"></p>
      <h2 for="out">Interpretation</h2>
      <p type="text" id="out" name="out"></p>
      <h2 for="bitarray">
        NLC Encoded
        <button onclick="copyBitArray()">Copy</button>
      </h2>
      <p type="text" id="bitarray" name="bitarray"></p>
      <h2 for="bitarray">ASCII Encoded</h2>
      <p type="text" id="ascii" name="ascii"></p>
    </div>
    <label for="dataAnalysis" class="dataAnalysis">For data analysis</label>
    <div id="dataAnalysis" class="dataAnalysis">
      <button onclick="copyWords()">Copy</button>
    </div>
    <script src="./script.js"></script>
  </main>
</body>
</html>
```

./progressive-compute/script.js

```
// Letters sorted by frequency on https://en.wikipedia.org/wiki/Letter_frequency
const specialIDs = [ "e", "t", "a", "o", "n", "r", "i", "s", "h", "d", "l", "f", "c", "m",
"u", "g", "y", "p", "w", "b", "v", "k", "j", "x", "z", "q", "0", "1", "2", "3", "4", "5", "6",
"7", "8", "9", "�" ]
const XON = "␑" // Uppercase control (ASCII Device Control One (XON)) is set as a constant,
because ␑ is difficult to type and not monospaced
const punctuation = [
    ".",
    ",",
    "-",
    "'",
    "\"",
    "$",
    "(",
    ")",
    ":",
    "?",
    "/",
    "!",
    ";",
    "\n",
    " ",
    XON,
]
let encoded = ""
let decoded = []
let formattedArray = []
let formattedBitArray = []

const maxRatio = 16 // Calculated manually (dependent on dataset)
const blueThreshold = 2 // Aesthetics

let brackets = true
let renderXON = true

let words = []

let wordModelGlobal = ""

let step = -2

let wordModel
let og
let idArray
let bitArray
let simplicity
let punctuationOrder
let raw
let rawArray
let sp
let i
let punct

let speed = 10

let asciiOut

function setup() {
    changeWordModel()
    toggleBrackets()
}

function changeWordModel() {
    wordModelGlobal = document.getElementById("wordModel-select").value
    words = eval(wordModelGlobal)
}

function toggleBrackets() {
    brackets = document.getElementById("toggle-brackets").checked
}

function copyBitArray() {
    if (brackets) navigator.clipboard.writeText(formattedBitArray.join(""));
    if (!brackets) navigator.clipboard.writeText(encoded);
}

function toggleXON() {
    renderXON = document.getElementById("toggle-xon").checked
    refresh()
}

function startCompute() {
    step = 0
    asciiOut = ascii2bin(document.getElementById("og").value)
}

function draw() {
    for(let i = 0; i < speed; i++) refresh()
}

function copyWords() {
    let wordRatios = []
    for (let i = 0; i < 10; i++) {
        wordRatios.push([words[i], words[i].length * 2, Math.ceil(dec2bin(i +
punctuation.length + specialIDs.length, 0).length / 4)].join("\t"))
    }
    navigator.clipboard.writeText(wordRatios.join("\n"));
}

let i0
let i1
let i2
let i3

function refresh() {
    if (step == 0) {
        i0 = 0
        i1 = 0
        i2 = 0
        i3 = 0

        wordModel = wordModelGlobal
        idArray = []
        formattedArray = []
        formattedBitArray = []
        bitArray = []

        simplicity = false
        punctuationOrder = 0
        og = document.getElementById("og").value

        // Convert og to rawArray (seperate by space and punctuation)
        raw = og.replaceAll(/([A-Z])/g, "␑$1").toLowerCase() // Replace uppercase characters
with upppercase control characters
        rawArray = []
        sp = 0
        i = 0
        punct = ""

        step++
    }

    if (step == 1) {
        if (i0 < raw.length) {
            while (!punctuation.includes(raw.charAt(i0))) { // Increase i until a punctuation
is reached
                i0++
                if (i0 >= raw.length) break // Protects against freezing
            }
            punct = raw.charAt(i0)
            if (punct == " ") {
                if (punctuation.includes(raw.charAt(i0 + 1))) {
                    rawArray.push(raw.substring(sp, i0), " ") // Spaces are not ignored if a
punctuation is after the space (He said, "Hello World.")
                } else {
                    rawArray.push(raw.substring(sp, i0)) // Spaces are ignored
                }
            } else {
                rawArray.push(raw.substring(sp, i0), punct) // Punctuation with no preceding
spaces are added
            }
            i0++
            while (punctuation.includes(raw.charAt(i0))) { // Accounts for all punctuation
after 1st punctuation
                punct = raw.charAt(i0)
                rawArray.push(punct)
                i0++
            }
            sp = i0
        } else step++
    }

    if (step == 2) {
        // Assign identifiers
        if (i1 < rawArray.length) {
            if (punctuation.includes(rawArray[i1])) {
                // Check punctuation
                idArray.push(punctuation.indexOf(rawArray[i1]))
                if (brackets) formattedArray.push(`<span style="color:
#ffd700;">[${rawArray[i1].replace("\n", "<br>")}]</span>`)
                if (!brackets) formattedArray.push(`<span style="color:
#ffd700;">${rawArray[i1].replace("\n", "<br>").replace(XON, renderXON ? XON : "")}</span>`)
            } else if (words.includes(rawArray[i1])) {
                // Check words
                idArray.push(words.indexOf(rawArray[i1]) + punctuation.length +
specialIDs.length)

                if (brackets) formattedArray.push(`<span style="color:
rgb(0,255,0);">(${rawArray[i1]})</span>${!(punctuation.includes(rawArray[i1 + 1])) &&
rawArray[i1 + 1] != "" ? " " : ""}`) // Adds text to formattedArray, along with whitespace if
there is not following puntuation
                if (!brackets) formattedArray.push(`<span style="color:
rgb(0,255,0);">${rawArray[i1]}</span>${!(punctuation.includes(rawArray[i1 + 1])) &&
rawArray[i1 + 1] != "" ? " " : ""}`)

            } else if (rawArray[i1] != "") {
                // Create words from letters, numbers, and unknown character
                for (let j = 0; j < rawArray[i1].length; j++) {
                    idArray.push(specialIDs.indexOf(rawArray[i1][j]) + punctuation.length)
                    if (!specialIDs.includes(rawArray[i1][j]))
idArray.push(specialIDs.indexOf("�") + punctuation.length)
                }
                if (brackets) formattedArray.push(`<span style="color: rgb(255, 0,
0);">{${rawArray[i1]}}</span>`)
                if (!brackets) formattedArray.push(`<span style="color: rgb(255, 0,
0);">${rawArray[i1]}</span>`)
                if (!punctuation.includes(rawArray[i1 + 1])) {
                    idArray.push(punctuation.indexOf(" "))
                    formattedArray.push(" ")
                }
            }
            i1++
        } else step++
    }

    if (step == 3) {
        // Create binary datastream

        // Control (could be useful in future iterations, disabled for now)
        //bitArray.push(
        //  simplicity ? 1 : 0,
        //  wordModel == "web333k" ? 0 : 1,
        //  punctuationOrder
        //)
        //if (brackets) formattedBitArray.push(simplicity ? 1 : 0, wordModel == "web333k" ? 0
: 1, punctuationOrder, " -    ")
        step++
    }

    if (step == 4) {
        // Indexes
        if (i2 < idArray.length) {
            bitArray.push(dec2bin(Math.ceil(dec2bin(idArray[i2], 0).length / 4), 3))
            if (brackets) formattedBitArray.push(dec2bin(Math.ceil(dec2bin(idArray[i2],
0).length / 4), 3), " ")
            i2++
        } else step++
    }

    if (step == 5) {
        bitArray.push("000")
        if (brackets) formattedBitArray.push(" (000) - ")
        step++
    }

    if (step == 6) {
        if (i3 < idArray.length) {
            /*
            dec2bin(310,0) =>
            "100110110" length = 9 (lead to the nearest 4-bit boundary: should be 12)
            4*floor(x/4+1)
            9 => 12
            dec2bin(310,12) => "000100110110" length = 9
            */
            bitArray.push(dec2bin(idArray[i3], 4 * Math.ceil(dec2bin(idArray[i3], 0).length /
4)))
            if (brackets) {
                for (let j = 0; j < bitArray[bitArray.length - 1].length / 4; j++) {
                    formattedBitArray.push(bitArray[bitArray.length - 1].substring(j * 4, j *
4 + 4), " ")
                }
                if (i3 + 1 < idArray.length) formattedBitArray.push(" | ")
            }
            i3++
        } else step++


    }

    if (step == 7) {
        decode()
        step = -1
    }

    if (step != -2) {
        encoded = bitArray.join("")
        if (brackets) document.getElementById("bitarray").innerHTML =
formattedBitArray.join("")
        if (!brackets) document.getElementById("bitarray").innerHTML = encoded
        document.getElementById("outin").innerHTML = formattedArray.join("")
        document.getElementById("out").innerHTML = decoded
        document.getElementById("ascii").innerHTML = asciiOut

        document.getElementById("stats").innerHTML =
            "<br>NLC Size - " + encoded.length +
            "<br>ASCII Size - " + asciiOut.length +
            "<br>Ratio - " + asciiOut.length / encoded.length + " : 1"
        if (step == -1) step = -2
    }
}

function decode() {
    let indexArray = []
    let idArray = []
    let idLookup = [...punctuation, ...specialIDs, ...words]
    decoded = []

    // Control
    //let simplicity = encoded[0] == 0 ? false : true
    //if (!simplicity) {
    //  let wordModel = encoded[1] == 0 ? "web333k" : "wikipedia1k"
    //  let punctuationOrder = encoded[2]
    //}

    // Indexes
    for (let i = 0; true; i++) {
        if (encoded.substring(3 * i, 3 * i + 3) == "000") break
        indexArray.push(bin2dec(encoded.substring(3 * i, 3 * i + 3)))
    }

    // IDs
    let j = indexArray.length * 3
    for (let i = 0; i < indexArray.length; i++) {
        idArray.push(bin2dec(encoded.substring(j + 3, j + 4 * indexArray[i] + 3)))
        j = j + 4 * indexArray[i]
    }

    // Convert IDs to content
    for (let i = 0; i < idArray.length; i++) {
        decoded.push(idLookup[idArray[i]].replace("\n", "<br>"))

        // If word followed by word, place space
        if (words.includes(idLookup[idArray[i]]) && (words.includes(idLookup[idArray[i + 1]])
|| specialIDs.includes(idLookup[idArray[i+1]]))) decoded.push(" ")
    }

    // Create uppercases
    decoded = decoded.join("").split("") // Turn into an array split by characters
    for(let i = 0; i < decoded.length; i++) {
        if (decoded[i] == XON) {
            decoded.splice(i,1)
            decoded[i] = decoded[i].toUpperCase()
        }
    }
    decoded = decoded.join("")
}

function dec2bin(dec, length) {
    // returns binary of decimal. dec is decimal input. length determines necessary leading
zeroes
    // ex. dec2bin(19, 8) => 00010011 dec2bin(19,0) => 10011
    let bin = ""
    while (true) {
        bin = dec % 2 + bin
        dec = Math.floor(dec / 2)
        if (dec == 0) break
    }
    let binlen = bin.length
    for (let i = 0; i < length - binlen; i++) {
        bin = 0 + bin
    }
    return (bin)
}

function bin2dec(bin) {
    let binarray = bin.split("")
    let dec = 0
    for (let i = 0; i < binarray.length; i++) {
        dec += Math.pow(2, i) * binarray[binarray.length - i - 1]
    }
    return (dec)
}


function ascii2bin(input) {
    // Credit: https://www.geeksforgeeks.org/javascript-program-to-convert-a-string-to-binary/
    let binaryResult = "";

    for (let i = 0; i < input.length; i++) {
        const charCode = input.charCodeAt(i);
        let binaryValue = "";
        for (let j = 7; j >= 0; j--) {
            binaryValue += (charCode >> j) & 1;
        }

        binaryResult += binaryValue;
    }

    return binaryResult.trim();
}
```

Iteration 3

./index.html

```
<!DOCTYPE html>
<html>
<head>
  <script src="./p5.js"></script>
  <script src="webSampleSentences10k.js"></script> <!--Use eng-uk_web-public_2018_300K-
sentences.js for sentence testing-->
  <!--<script src="eng-uk_web-public_2018_300K-sentences.js"></script> <!---->
  <script src="./x16web.js"></script>
  <script src="./web-15.js"></script>
  <title>Natural Language Encoder</title>

  <style>
    body {
      background-color: #181a1b;
      color: white
    }

    .input-container {
      padding-right: 7px;
      margin-bottom: 50px;
    }

    textarea {
      width: 100%;
      height: 100px;
      resize: vertical;
      font-size: medium;
    }

    p {
      font-family: monospace;
      font-size: medium;
      word-wrap: break-word;
      white-space: pre-wrap;
    }

    select {
      padding-bottom: 0px;
      padding-top: 0px;
    }

    #dataset-stats {
      margin-top: 0px;
      margin-bottom: 0px;
    }

    .indented {
      border-left-color: white;
      border-left-width: 1px;
      border-left-style: solid;
      padding-left: 8px;
    }

    h2 {
      font-size: 1.25em;
      margin-bottom: 0;
    }
```

```css
#stats {
  border-left-color: rgb(0, 255, 0);
}

a, a:visited {
  color: #3391ff;
}

button, input, textarea, select {
  background-color: #2b2a33;
  color: white;
  border-color: #8f8f9d;
  border-radius: 5px;
}

.tooltip {
  position: relative;
  display: inline-block;
}

.tooltip .tooltiptext {
  visibility: hidden;
  width: max-content;
  background-color: #2b2a33;
  color: #fff;
  text-align: left;
  padding: 5px 0;
  border-radius: 6px;

  position: absolute;
  z-index: 1;
}

.tooltip:hover .tooltiptext {
  visibility: visible;
}

.tooltip .tooltiptext {
  bottom: 100%;
  left: 50%;
  margin-left: -50%;
}

.flex-container {
  display: flex;
  justify-content: space-between;
}

#testing-menu {
  border-right-color: white;
  border-right-width: 1px;
  border-right-style: solid;
  padding-right: 8px;
  display: flex;
  flex-direction: column;
  align-items: end;
}

#testing-menu * {
  margin-top: 4px;
```

```
    }

    .dataAnalysis {
      visibility: visible;
    }

  </style>
  <link rel="stylesheet" href="darkmode.css">
</head>
<body onresize="changeSettings()">
  <main>
    <a href="./decode">Natural Language Decoder</a><br>
    <h1>Natural Language Encoder v3</h1>
    <div class="flex-container">
      <div id="model-selector" class="indented">
        <label for="dataset-select">Lexeme Model:</label>
        <select id="dataset-select" name="dataset-select" onchange="changeSettings()">
          <option value="web15" selected="true">web15</option>
        </select>
        <p id="dataset-stats"></p>
      </div>
      <div id="testing-menu">
        <label for="testing-menu">Testing Menu</label>
        <button onclick="commenceTesting()">Commence Sentence Tests</button>
        <button onclick="copyResults()">Copy Sentence Test Results</button>
        <table id="test-stats"></table>
      </div>
    </div>

    <p id="stats" class="indented"></p>

    <div class="input-container">
      <label for="og">Natural Language Input</label><br>
      <label for="live-encode">Encode Live</label>
      <input type="checkbox" id="live-encode" name="live-encode" checked="true"
onchange="changeSettings()">
      <button onclick="encodeInput()">Encode</button>
      <button onclick="newSentence()">Random Sentence</button>
      <textarea id="og" name="og" onkeyup="encodeLive()"></textarea>
    </div>
    <div class="output-container">
      <h2 for="outin">Original</h2>
      <label for="toggle-brackets">Render Brackets</label>
      <input type="checkbox" id="toggle-brackets" name="toggle-brackets" checked="true"
onchange="changeSettings()">
      <p type="text" id="outin" name="outin"></p>
      <h2 for="out">Interpretation</h2>
      <p type="text" id="out" name="out"></p>
      <h2 for="bitarray">
        NLC Encoded
        <button onclick="copyBitArray()">Copy</button><br>
      </h2>
      <label for="toggle-force-detail">Force Detailed View</label>
      <input type="checkbox" id="toggle-force-detail" name="toggle-force-detail"
onchange="changeSettings()">
      <p type="text" id="bitarray" name="bitarray"></p>
      <h2 for="bitarray">ASCII Encoded</h2>
      <p type="text" id="ascii" name="ascii"></p>
    </div>
    <label for="dataAnalysis" class="dataAnalysis">For data analysis</label>
    <div id="dataAnalysis" class="dataAnalysis">
```

```
      <button onclick="analyseIndexingSchemes()">Analyse Indexing Schemes</button>
      <button onclick="preconfigCopy()">Copy Preconfigured Dataset</button>
      <!--<button onclick="copyErrorWords()">Copy Unused Words</button>-->
    </div>
    <script src="./script.js"></script>
  </main>
</body>
</html>
```

./script.js

```
/*
Author: Brian Heers
ISEF 2025 NLCv3 script.js

 * An implementation of Natural Language Compression (NLC), a method to encoding and decoding
natural language as binary in JavaScript.
 * A demo is available at isef.heers.net/v3.

*/

// Special characters not monospaced in VS Code
const XON = "␑" // Space override (ASCII Device Control One (XON)) is set as a constant,
because ␑ is difficult to type and not monospaced
const DC2 = "␒" // No space override (ASCII Device Control Two (DC2)) is set as a constant,
because ␒ is difficult to type and not monospaced
const SUB = "�" // Unknown substitute character to substitute any unencodable lexemes

// Global settings
let liveEncode
let wordModelGlobal
let brackets
let forceDetailed
let encoded

// Testing for single-letter lexeme placement (errorWord) (now deprecated)
let errorWords = []
let totalWords = 0

// Letters sorted by frequency on https://en.wikipedia.org/wiki/Letter_frequency
let freqLetters = ["e", "t", "a", "o", "n", "r", "i", "s", "h", "d", "l", "f", "c", "m", "u",
"g", "y", "p", "w", "b", "v", "k", "j", "x", "z", "q"]

//document.getElementById("og").value = "" // Debug a specific string

// Global scope variables
let formattedArray = []

changeSettings()

/** Autofills the textarea with a randomly selected sentence from the webSentences dataset
(either 10k or 300k, however it is set in the HTML) */
function newSentence() {
    document.getElementById("og").value = sampleSentences[Math.floor(Math.random() *
sampleSentences.length)]//.toLowerCase()
    encodeLive()
}

/** Updates all settings whenever a setting is changed on the HTML, and encodes the input with
the new settings if liveEncode is enabled*/
function changeSettings() {
    brackets = document.getElementById("toggle-brackets").checked
    liveEncode = document.getElementById("live-encode").checked
    forceDetailed = document.getElementById("toggle-force-detail").checked
    document.getElementById("dataset-stats").innerHTML = [
        `Lexemes:                    (${web15.indexed.lexemes.slice(0, 8).join("), (")}),
..., (${web15.indexed.lexemes[web15.lexemes.length - 1]})`,
        `Sorted by Ratio:            (${web15.lexemes[0]}), ...,
(${web15.lexemes[web15.lexemes.length - 1]})`,
        `Elements:                     ${web15.lexemes.length}
(2^${Math.log2(web15.lexemes.length)})`,
        `Total Bits ASCII:             ${web15.totalBitsASCII}`,
        `Simulated Total Bits NLC:     ${web15.totalBitsNLC}`,
        `Simulated Ratio:              ${web15.simulatedRatio}`,
        `Indexing Scheme:              [ ${web15.indexingScheme.join(", ")} ]`,
        `Index Assignment Bit Length: ${web15.indexAssignmentBitLength}`,
    ].join("\n")
    encodeLive()
}

/** Decodes NLCv3 binary into natural language. Takes the encoded binary and dataset to decode
with. The datasets must match for successful encoding.
 * @param {string} encoded - NLCv3 binary
 * @param {object} dataset - NLCv3 dataset
*/
function decode(encoded, dataset) {
    //Seperate index assignments until the sum of their indexes and the index is the length of
encoded
    let indexArray = []
    let i = 0
    let j = dataset.indexAssignmentBitLength
    while (sum(...indexArray, i) < encoded.length) {
        indexArray.push(dataset.indexingScheme[bin2dec(encoded.substring(i, j))])
        i += dataset.indexAssignmentBitLength
        j += dataset.indexAssignmentBitLength
    }

    // Get lexemes
    let lexemeArray = []
    j = i
    for (let k = 0; k < indexArray.length; k++) {
        j += indexArray[k]
        lexemeArray.push(web15.indexed.lexemes[(bin2dec(encoded.substring(i, j)))])
        i += indexArray[k]
    }

    // Spacing
    let output = ""
    for (let i = 0; i < lexemeArray.length; i++) {
        const pre = lexemeArray[i - 1] // Previous
        const cur = lexemeArray[i] // Current
        const nex = lexemeArray[i + 1] // Next
        function ns(lex) { return dataset.spaceDefaults.noSpaces.includes(lex) } // No space
        function rs(lex) { return dataset.spaceDefaults.rightSpace.includes(lex) } // Right
space
        function ls(lex) { return dataset.spaceDefaults.leftSpace.includes(lex) } // Left
space

        if (ns(cur) || rs(cur)) {
            output += cur
        } else if (ns(pre)) {
            output.slice(0, -1)
            output += cur
        } else if (!ls(pre)) {
            output += " " + cur
        } else {
            output += cur
        }
    }
    return output.trim().replaceAll(XON, " ").replaceAll(DC2, "")

    /** Calculates the sum of its arguments. Expects a spreaded array (...[]), NOT a literal
array ([]) */
    function sum() {
        let sum = 0
        for (let i = 0; i < arguments.length; i++) {
            sum += arguments[i]
        }
        return sum
    }
}

/** If liveEncode is active, will encode the input textarea */
function encodeLive() {
    if (liveEncode) {
        encodeInput()
    }
}

/** Encodes the input textarea, and updates all relevent HTML elements. Also returns
statistics of the encoding. */
function encodeInput() {
    console.clear()
    formattedArray = []

    encoded = encode(document.getElementById("og").value, web15, brackets)
    ascii = ascii2bin(document.getElementById("og").value)

    document.getElementById("outin").innerHTML = formattedArray.join("").trim()

    if (brackets) {
        document.getElementById("bitarray").innerHTML = encoded.bracketed
        document.getElementById("ascii").innerHTML = ascii.spaced
    } else {
        document.getElementById("bitarray").innerHTML = encoded.default
        document.getElementById("ascii").innerHTML = ascii.default
    }

    document.getElementById("stats").innerHTML = [
        `NLC Bits:   ${encoded.default.length}`,
        `ASCII Bits: ${ascii.default.length} (${ascii.default.length / 8} * 8)`,
        `Ratio:      ${ascii.default.length / encoded.default.length} : 1`
    ].join("\n")

    document.getElementById("out").innerHTML = decode(encoded.default, web15)

    return {
        input: document.getElementById("og").value,
        bitsNLC: encoded.default.length,
        bitsASCII: ascii.default.length,
        ratio: ascii.default.length / encoded.default.length,
        output: document.getElementById("out").innerHTML
    }
}

/** Encodes the given input, using the specified dataset
 * @param {string} inputString String of natural language, to be encoded
 * @param {Dataset} dataset NLCv3 dataset
 * @param {Boolean} brackets Whether or not to calculate and return a bracketed output
*/
```

```
function encode(inputString, dataset, brackets = false) {
    let lexemeArray = optimizeArray(splitIntoArray(inputString))
    let idArray = createIdArray(lexemeArray, dataset)
    let bitArray = createBitArray(idArray, dataset)

    // --- For errorWord testing ---
    /*
    let testArray = [...lexemeArray]
    for (let i = 0; i < testArray.length; i++) {
        if (testArray[i][0] == " ") {
            testArray.splice(i, 1)
        }
    }
    totalWords += testArray.length
    */
    // --- For errorWord testing ---

    renderColored(lexemeArray, dataset)

    renderTooltips(dataset, bitArray, lexemeArray)

    // Detailed and Undetailed Bracket View
    if (brackets) {
        var formattedBrackets = renderBracketed(bitArray, dataset)
    }

    return {
        default: bitArray.indexes.join("") + bitArray.IDs.join(""),
        bracketed: formattedBrackets
    }

    // Internal-scope functions: renderColored, renderTooltips

    /** Takes a string and an HTML color string to return a colored span tag
       * @param {string} string The text to display in the innerHTML
       * @param {string} colorString The color to display the text as in HTML format (e.g.
"rgb(102, 15, 232)")
    */
    function htmlColor(string, colorString) {
        return `<span style="color: ${colorString};">${string}</span>`
    }

    /** Create the output shown in NLC Encoded when brackets are enabled */
    function renderBracketed(bitArray, dataset) {
        let bitArrayBracketed = bitArray
        let header = [
            "              <i>i</i>: ",
            "         Binary: ",
            "        Decimal: ",
            "Index / Lexeme: "
        ]
        let formattedBrackets = [...header] // ["", "", "", "",]

        // --- Index half ---
        for (let i = 0; i < bitArray.indexes.length; i++) {
            // Values to be assigned to headers
            let values = [
                i,
                bitArray.indexes[i],
                bin2dec(bitArray.indexes[i]),
                dataset.indexingScheme[bin2dec(bitArray.indexes[i])]
            ]

            // Pad all elements to be flushed left
            let longestElement = 0
            for (let j = 0; j < values.length; j++) {
                longestElement = longestElement < values[j].length ? values[j].length :
longestElement
            }
            for (let j = 0; j < values.length; j++) {
                formattedBrackets[j] += String(values[j]).padEnd(longestElement + 1)
            }
        }

        // --- ID half ---
        // Index-ID separators
        formattedBrackets[0] += "  "
        formattedBrackets[1] += "- "
        formattedBrackets[2] += "  "
        formattedBrackets[3] += "  "

        for (let i = 0; i < bitArray.IDs.length; i++) {
            // Values to be assigned to headers
            let values = [
                i,
                bitArray.IDs[i] + (i < (bitArray.IDs.length - 1) ? " |" : ""),
                bin2dec(bitArray.IDs[i]),
                dataset.indexed.lexemes[bin2dec(bitArray.IDs[i])].replace("\n", "\\n")
            ]

            // Pad all elements to be flushed left
            let longestElement = 0
            for (let j = 0; j < values.length; j++) {
                longestElement = longestElement < values[j].length ? values[j].length :
longestElement
            }
            for (let j = 0; j < values.length; j++) {
                formattedBrackets[j] += String(values[j]).padEnd(longestElement + 1)
            }
        }

        if (formattedBrackets[0].length > screen.width / 9 && !forceDetailed) {
            // If there is too much text for the screen and details aren't being forced,
fallback to old bracketed mode: 110 100 - 010000101111101 | 0100001101
            formattedBrackets = bitArrayBracketed.indexes.join(" ") + " - " +
bitArray.IDs.join(" | ")
        } else {
            formattedBrackets[1] = htmlColor(formattedBrackets[1], "rgb(192, 255, 192)")
            formattedBrackets = formattedBrackets.join("\n")
        }
        return formattedBrackets
    }

    /** Create tooltips for each lexeme. Directly edits formattedArray. */
    function renderTooltips(dataset, bitArray, lexemeArray) {
        //<span class="tooltip">Hover over me<span class="tooltiptext">Tooltip
text</span></span>
        if (brackets) {
            let offset = 0
            for (let i = 0; i < formattedArray.length; i++) {
                let j = i - offset
                formattedArray[i] = `<span class="tooltip">${formattedArray[i]}<span
class="tooltiptext">${[
                    `Lexeme:        <b>${lexemeArray[i].replace(XON, "XON (Override: Add
Space)").replace(DC2, "DC2 (Override: Remove Space)").replace("\n", "\\n")}</b>`,
                    `<i>i</i>:             ${j}`,
                    `Length:        ${lexemeArray[i].length}`,
                    `Ranking:       ${dataset.indexed.lexemes.indexOf(lexemeArray[i])}`,
                    `Ratio Ranking: ${dataset.lexemes.indexOf(lexemeArray[i])}`,
                    `Index:         ${bitArray.indexes[j]} ->
${dataset.indexingScheme[bin2dec(bitArray.indexes[j])]}`,
                    `ASCII Bits:    ${lexemeArray[i].length * 8}`,
                    `NLC Bits:      ${(bitArray.IDs[j] + bitArray.indexes[j]).length}
(${bitArray.indexes[j]} + ${bitArray.IDs[j]})`,
                    `Ratio:         ${Math.round((lexemeArray[i].length * 8) /
(bitArray.IDs[j] + bitArray.indexes[j]).length * 100) / 100}`,
                ].join("\n")}</span></span>`
            }
        }
    }

    /** First step on formattedArray, brackets and colors the text */
    function renderColored(lexemeArray, dataset) {
        for (let i = 0; i < lexemeArray.length; i++) {
            if (lexemeArray[i] == XON || lexemeArray[i] == DC2) {
                //errorWords.push(XON)
                if (brackets) {
                    formattedArray.push(htmlColor(`[${lexemeArray[i]}]`, "#ffd700"))
                } else {
                    formattedArray.push(htmlColor(`${lexemeArray[i]}`, "#ffd700"))
                }
            } else {
                let pre = lexemeArray[i - 1] // Previous
                let cur = lexemeArray[i] // Current
                let nex = lexemeArray[i + 1] // Next
                function ns(lex) { return dataset.spaceDefaults.noSpaces.includes(lex) } // No
space
                function rs(lex) { return dataset.spaceDefaults.rightSpace.includes(lex) } //
Right space
                function ls(lex) { return dataset.spaceDefaults.leftSpace.includes(lex) } //
Left space
                if (brackets) {
                    formattedArray.push(htmlColor(`(${cur.replaceAll("\n", "\\n")})`, "rgb(0,
255, 0)"))
                } else {
                    let output = ""
                    if (ns(cur) || rs(cur)) {
                        output += cur
                    } else if (ns(pre)) {
                        output.slice(0, -1)
                        output += cur
                    } else if (i > 0 && !ls(pre)) {
                        output += " " + cur
                    } else {
                        output += cur
                    }
                    formattedArray.push(htmlColor(`${output.replaceAll("<br>", "\\n")}`,
"rgb(0, 255, 0)"))
                }
            }
        }
    }
    /** Takes an already created idArray and dataset and returns an object of arrays of binary
IDs and indexes
     * @param idArray Generated by createIdArray()
     * @param dataset NLCv3 Dataset
    */
    function createBitArray(idArray, dataset) {
        let indexBitArray = []
        let idBitArray = []
        for (let i = 0; i < idArray.length; i++) {
            let indexAssignment
            for (let j = 0; j < dataset.indexingScheme.length; j++) {
                if (dec2bin(idArray[i], 0).length <= dataset.indexingScheme[j]) {
                    indexAssignment = dataset.indexingScheme[j]
                    break
                }
            }
            let binaryID = dec2bin(idArray[i], indexAssignment)
            indexBitArray.push(dec2bin(dataset.indexingScheme.indexOf(indexAssignment),
dataset.indexAssignmentBitLength))
            idBitArray.push(binaryID)
        }

        return {
            IDs: idBitArray,
            indexes: indexBitArray
        }
    }

    /** Takes an already created lexemeArray and dataset and returns an idArray. In charge of
assigning IDs to lexemes and spacing handling as defined in Iteration 3 Write Up
     * @param lexemeArray Generated by optimizeArray(splitArray(inputString))
     * @param dataset NLCv3 Dataset
    */
    function createIdArray(lexemeArray, dataset) {
        let idArray = []
        for (let i = 0; i < lexemeArray.length; i++) {
            let dpr = lexemeArray[i - 2] // Double Previous
            let pre = lexemeArray[i - 1] // Previous
            let cur = lexemeArray[i] // Current
            let nex = lexemeArray[i + 1] // Next
            function ns(lex) { return dataset.spaceDefaults.noSpaces.includes(lex) } // No
space
            function rs(lex) { return dataset.spaceDefaults.rightSpace.includes(lex) } //
Right space
            function ls(lex) { return dataset.spaceDefaults.leftSpace.includes(lex) } // Left
space

            if (ns(cur)) {
                if (pre == " " && !ns(dpr)) {
                    lexemeArray.splice(i - 1, 0, XON)
                    i++
                }
                if (nex == " ") {
                    lexemeArray.splice(i + 1, 0, XON)
                    i++
                }
            } else if (rs(cur) && nex != " " && i + 1 < lexemeArray.length) {
                lexemeArray.splice(i + 1, 0, DC2)
                i++
            } else if (pre == DC2 && ls(dpr)) {
                lexemeArray.splice(i - 1, 1)
            } else if (cur != " " && nex != " " && i + 1 < lexemeArray.length && ((!rs(nex) &&
!ns(nex)) || ls(nex))) {
                lexemeArray.splice(i + 1, 0, DC2)
                i++
            } else if (cur == " " && ns(nex)) {
                lexemeArray.splice(i, 1, XON)
            } else if (cur == " " && (rs(nex))) {
                lexemeArray.splice(i, 1, XON)
            } else if (ls(cur) && nex == " ") {
                lexemeArray.splice(i + 1, 0, XON)
                i++
            }
        }


        while (lexemeArray.includes(" ")) {
            lexemeArray.splice(lexemeArray.indexOf(" "), 1)
        }

        for (let i = 0; i < lexemeArray.length; i++) {
            let id = dataset.indexed.lexemes.indexOf(lexemeArray[i])
            if (id == -1) {
                id = dataset.indexed.lexemes.indexOf(SUB)
                console.log("Unencodable")
            }
            idArray.push(id)
        }
        return idArray
    }

    /** Implements Lexeme Determination by Highest-Ratio Substringing (LDHR) by iterating from
the highest ratio lexemes to the lowest and progressively splitting the substrings into an
array of lexemes. View the LDHR section in v3 Write-Up for more information
     * @param {string} input
    */
    function splitIntoArray(input) {
        function u(unencoded) {
            return [unencoded, "unencoded"]
        }
        function e(encoded) {
            return [encoded, "encoded"]
        }

        /** Searches string for the first instance of stringCondition, and splits it into an
array
         * @param {string} string String to search through
         * @param {string} stringCondition String to search for
        */
        function splitFirstInstance(string, stringCondition) {
            return [
                string.substring(0, string.indexOf(stringCondition)),
                string.substring(string.indexOf(stringCondition) + stringCondition.length),
            ]
        }

        /** If this is the second pass (resplitting after being optimized), already consider
text \\within backslashes\\ to be encoded */
        function preSplit(input) {
            const regex = /\\(.*?)\\/g
            let result = []
            let lastIndex = 0

            input.replace(regex, (match, p1, offset) => {
                if (offset > lastIndex) {
                    result.push(u(input.slice(lastIndex, offset)))
                }
                result.push(e(p1))
                lastIndex = offset + match.length
            })

            if (lastIndex < input.length) {
                result.push(u(input.slice(lastIndex)))
            }

            return result
        }

        let inputArray = []
        if (input.includes("\\")) {
            inputArray.push(...preSplit(input))
        } else {
            inputArray.push(u(input))
        }

        // Splits the string into lexemes by going through the highest ratio words first
        for (let i = 0; i < web15.ratios.length; i++) {
            for (let j = 0; j < inputArray.length; j++) {
                if (inputArray[j][1] == "unencoded" &&
inputArray[j][0].includes(web15.lexemes[i])) {
                    let splitArray = splitFirstInstance(inputArray[j][0], web15.lexemes[i])
                    inputArray.splice(j, 1, u(splitArray[0]), e(web15.lexemes[i]),
u(splitArray[1]))
                    j = 0
                }
            }
        }

        let outputArray = []
        for (let i = 0; i < inputArray.length; i++) {
            if (inputArray[i][0] != "") {
                if (inputArray[i][1] == "unencoded" && inputArray[i][0] != " ") {
                    console.log(inputArray[i][0])
                    errorWords.push(inputArray[i][0].trim())
                } else {
                    outputArray.push(inputArray[i][0])
                }
            }
        }
        return outputArray
    }

    /** Some words require optimization: suppose "tone" is being encoded. (tone) is encoded in
18 bits and has a ratio of 1.78, but (to) has a ratio of 2.76, so in the first step of LDHR,
[tone] will be split to (to)[ne], and then (to)(ne), which is encoded in 24 bits (001 110 -
101 | 110000111100100), and (ne) alone is 18 bits, which was as much as (tone). optimizeArray
iterates through each word and if its LDHR splitting is less efficient than encoding the word
as a single word, it marks it to be encoded as a single word. This way, [tone] will first be
split to (to)[ne], and then tested if (tone) exists and if it is more efficient, then it will
mark (tone) as \\tone\\, meaning it is to be split by itself as it passes everything back to
the splitIntoArray function. */
    function optimizeArray(inputArray) {
        /** Searches string for the first instance of stringCondition, and splits it into an
array, but includes the stringCondition in the output
         * @param {string} string String to search through
         * @param {string} stringCondition String to search for
        */
        function splitFirstInstanceInclusive(string, stringCondition) {
            toReturn = []

            if (string.substring(0, string.indexOf(stringCondition)) != "")
toReturn.push(string.substring(0, string.indexOf(stringCondition)))
            toReturn.push(stringCondition)
            if (string.substring(string.indexOf(stringCondition) + stringCondition.length) !=
"") toReturn.push(string.substring(string.indexOf(stringCondition) + stringCondition.length))

            return toReturn
        }

        // First split everything by words (characters seperated by spaces)
        let wordArray = inputArray.join("").split(" ")
        for (let i = 0; i < wordArray.length; i++) {
            let testedWord = wordArray[i]
            if (web15.lexemes.includes(testedWord)) {
                // If the word is a lexeme and is more efficiently encoded as a word than
through LDHR, it is \\slashMarked\\
                if (simulateLexeme(web15, testedWord).ratio > simulateLexeme(web15,
...splitIntoArray(testedWord)).ratio) {
                    wordArray = slashMark(testedWord)
                    // (it.) evaluated to \([^\\]||$)(it.)([^\\]||^)\g and . is a wildcard!
This bug took 3 hours to discover...
                }
            } else if (testedWord != "" && !testedWord.includes("\\")) {
                // Otherwise, test every possible configuration of up to three lexemes and
pick the most efficient split
                let errorWord = testedWord
                let possibleSplits = []
                // Console logs to show progress, because DOM elements cannot be editted
during computing
                console.log(`Optimizing: (${errorWord})`)
                let detailedLog = false
                if (errorWord.length > 16) {
                    // If the lexeme is significantly long, optimization can take several
seconds and require more frequent logs to keep the browser from freezing and the user from
getting impatient
                    console.log(`Lexeme is ${errorWord.length} characters long,\nOptimization
progress:`)
                    detailedLog = true
                }
                let testedSubstring
                for (let j = 0; j < errorWord.length; j++) {
                    if (detailedLog) {
                        console.log(`${Math.round(100 * j / errorWord.length)}% Complete`)
                    }
                    for (let k = 0; k < errorWord.length; k++) {
                        testedSubstring = errorWord.substring(k, errorWord.length - j + k)
                        // Split and simulate
                        if (simulateLexeme(web15, ...splitFirstInstanceInclusive(errorWord,
testedSubstring)).ratio >= simulateLexeme(web15, ...splitIntoArray(errorWord)).ratio) {
                            possibleSplits.push({ split:
splitFirstInstanceInclusive(errorWord, testedSubstring), ratio: simulateLexeme(web15,
...splitFirstInstanceInclusive(errorWord, testedSubstring)).ratio })
```

```
                        }
                    }
                }
                let highestRatio = possibleSplits[0]
                for (let j = 0; j < possibleSplits.length; j++) {
                    highestRatio = possibleSplits[j].ratio > highestRatio.ratio ?
possibleSplits[j] : highestRatio
                }
                if (highestRatio != undefined) {
                    // Mark the highestRatio word with backslashes
                    let toReplace = highestRatio.split.join("")
                    let replaceWith = ""
                    for (let j = 0; j < highestRatio.split.length; j++) {
                        replaceWith += "\\" + highestRatio.split[j] + "\\"
                    }
                    wordArray = wordArray.join(" ").replaceAll(toReplace, replaceWith).split("
")
                }
            }
        }

        return splitIntoArray(wordArray.join(" "))

        /** Marks backslashes enveloping the markedWord using RegEx */
        function slashMark(markedWord) {
            return wordArray.join("
").replaceAll(eval(`/([^\\\\]{1}|^)(${markedWord.replaceAll(/([\^\$\\\.\*\+\?\(\)\[\]\{\}\|\/]
)/g, "\\$1")})([^\\\\]{1}|$)/g`), "$1\\$2\\$3").split(" ")
        }
    }

    /** A quick function to calculate the amount of bits lexeme(s) will require in NLC and
ASCII. Takes a dataset, followed by lexemes to encode. Do not pass an array; only pass spread
arrays.
     * @param dataset NLCv3 Dataset
     * @param {ArrayIterator} arguments Lexemes to simulate
    */
    function simulateLexeme(dataset) {
        // Takes dataset followed by lexeme(s) to simulate
        let bitsNLC = 0
        let bitsASCII = 0
        // Find which index works best for the word
        for (let i = 1; i < arguments.length; i++) {
            let lexeme = arguments[i]
            let smallestIndexAssignment
            for (let j = 0; j < dataset.indexed.indexingScheme.length; j++) {
                if (dataset.indexed.lexemes.indexOf(lexeme) < Math.pow(2,
dataset.indexed.indexingScheme[j])) {
                    if (!dataset.indexed.lexemes.includes(lexeme)) {
                        return NaN
                    }
                    smallestIndexAssignment = dataset.indexingScheme[j]
                    break
                }
            }
            bitsNLC += smallestIndexAssignment + dataset.indexAssignmentBitLength
            bitsASCII += lexeme.length * 8
        }
        return {
            bitsNLC: bitsNLC,
            bitsASCII: bitsASCII,
            ratio: bitsASCII / bitsNLC
        }
    }
}

/** Configuring the dataset means taking in a raw dataset (object in the form of {lexemes: [],
occurrences: []}), and returning an object containing the “simulation” and the indexingScheme
({...dataset, totalBitsASCII, totalBitsNLC, simulatedRatio, indexingScheme}.) To avoid doing
this calculation every time the program is refreshed or run, a preconfigured web-15.js dataset
can be created by taking the returned object from datasetConfig, and declaring it as let web15
= {…}.
 * @param highestIndex The amount of bits it would take to represent the whole dataset in
order to trim it to a power of 2
 * @param desiredIndexes desiredIndexes and indexAssignmentBitLength need to work together
(desiredIndexes <= 2^indexAssignmentBitLength)
 * @param indexAssignmentBitLength Flat rate of how many bits each word takes up in its index
 * @param {{lexemes: [], occurrences: []}} dataset
*/
function datasetConfig(highestIndex, desiredIndexes, indexAssignmentBitLength, dataset) {
    // specialIDs generated with errorWord testing, occurrences of 0 were made to 2 and sub
was made 1, DC2 was placed between XON and "a"
    let specialIDs = {
        chars: [
            XON, DC2, "a", "n", "r", "e", "s", "f", "i", "o", "m", "g", "t", "h", "l", "c",
"p", "d", "y", "b", "u", "w", "k", "x", "z", "v", "j", "q", "\n", SUB,
        ],
        occurrences: [
            97699, 70530, 43361, 39655, 16772, 13364, 12751, 7848, 7680, 6956, 5507, 5219,
4597, 4123, 3752, 2888, 2220, 1997, 1857, 1412, 1179, 427, 288, 121, 111, 74, 2, 2, 1, 0,
        ]
    }

    // Manually generated list of normal spacing rules for characters and punctuation
    let spaceDefaults = {
        noSpaces: [
            "'",
            "-",
            "\"",
            "\n",
            "n","r","e","f","i","o","m","g","t","h","l","c","p","d","y","b","u","w","k","x","z
","v","j","q",
            XON,
            DC2,
        ],
        rightSpace: [
            ".",
            ",",
            ":",
            ";",
            "!",
            ")",
            "%",
            "”",
            "’",
            "s",
            "?"
        ],
        leftSpace: [
            "“",
            "‘",
            "(",
            "$",
            "£",
            "€",
            "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P",
"Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z",
        ]
    }

    if (!(desiredIndexes <= Math.pow(2, indexAssignmentBitLength))) {
        console.error(`desiredIndexes and indexAssignmentBitLength need to work together
(desiredIndexes <= 2^indexAssignmentBitLength)\nCurrently: (${desiredIndexes} <= ${Math.pow(2,
indexAssignmentBitLength)}) --> False`)
        return
    }

    let totalOccurrences = 0
    let percentOfLanguage = []
    let percentOfWhole = []

    //Remove single letter lexemes
    for (let i = 0; i < dataset.lexemes.length; i++) {
        if (freqLetters.includes(dataset.lexemes[i])) {
            dataset.lexemes.splice(i, 1)
            dataset.occurrences.splice(i, 1)
        }
    }

    // Trim the dataset to align with highestIndex
    dataset.occurrences = dataset.occurrences.slice(0, Math.pow(2, highestIndex))
    dataset.lexemes = dataset.lexemes.slice(0, Math.pow(2, highestIndex))

    // Insert specialIDs
    dataset.lexemes.splice(-specialIDs.chars.length, specialIDs.chars.length,
...specialIDs.chars)
    dataset.occurrences.splice(-specialIDs.occurrences.length, specialIDs.occurrences.length,
...specialIDs.occurrences)
    console.log(dataset.lexemes)

    // Sort the dataset based on the occurrence column
    let combined0 = dataset.lexemes.map((lexeme, index) => ({
        lexeme,
        occurrence: dataset.occurrences[index]
    }));

    combined0.sort((a, b) => b.occurrence - a.occurrence);

    dataset.lexemes = combined0.map(item => item.lexeme);
    dataset.occurrences = combined0.map(item => item.occurrence);

    // Create a percentOfWhole array which shows how much of the dataset can be represented in
the above lexemes
    for (let i = 0; i < dataset.lexemes.length; i++) {
        totalOccurrences += dataset.occurrences[i]
    }
    for (let i = 0; i < dataset.lexemes.length; i++) {
        percentOfLanguage.push(dataset.occurrences[i] / totalOccurrences)
    }
    for (let i = 0; i < dataset.lexemes.length; i++) {
        let sum = 0
        for (let j = 0; j <= i; j++) {
            sum += percentOfLanguage[j]
        }
        percentOfWhole.push(sum)
    }

    // Create an indexing scheme using desiredIndexes and percentOfWhole
    let indexingScheme = []
    let numerator = 1
    for (let i = 0; i < percentOfWhole.length; i++) {
        if (percentOfWhole[i] >= numerator / desiredIndexes) {
            indexingScheme.push(Math.round(Math.log2(i)))
            numerator++
        }
    }
    indexingScheme.push("DIF")

    // Simulate whole dataset to test its theoretical efficiency and calculate a ratio for
each lexeme
    let totalBitsNLC = 0
    let totalBitsASCII = 0
    let wordRatios = []
    for (let i = 0; i < dataset.occurrences.length; i++) {
        // Find which index works best for the word
        let smallestIndexAssignment
        for (let j = 0; j < indexingScheme.length; j++) {
            if (i < Math.pow(2, indexingScheme[j])) {
                smallestIndexAssignment = indexingScheme[j]
                break
            }
        }

        let bitsNLC = (smallestIndexAssignment + indexAssignmentBitLength) *
dataset.occurrences[i] // Index assignment + content bits, all times the amount it occurs in
the corpus the data comes from
        let bitsASCII = dataset.lexemes[i].length * 8 * dataset.occurrences[i] // ASCII is
just 8 bits per character

        wordRatios.push(bitsASCII / bitsNLC)
        totalBitsNLC += bitsNLC
        totalBitsASCII += bitsASCII
    }
    let simulatedRatio = totalBitsASCII / totalBitsNLC
    dataset = {
        ...dataset,
        ratios: wordRatios
    }

    // Save this state as the indexedVersion before sorting by ratio
    let resultsIndexed = {
        ...dataset,
        totalBitsASCII: totalBitsASCII,
        totalBitsNLC: totalBitsNLC,
        simulatedRatio: simulatedRatio,
        indexingScheme: indexingScheme,
        indexAssignmentBitLength: indexAssignmentBitLength,
    }

    // Sort the dataset based on the ratio column
    let combined = dataset.lexemes.map((lexeme, index) => ({
        lexeme,
        occurrence: dataset.occurrences[index],
        ratio: wordRatios[index]
    }));

    combined.sort((a, b) => b.ratio - a.ratio);

    dataset.lexemes = combined.map(item => item.lexeme);
    dataset.occurrences = combined.map(item => item.occurrence);
    dataset.ratios = combined.map(item => item.ratio);

    let results = {
        ...dataset,
        totalBitsASCII: totalBitsASCII,
        totalBitsNLC: totalBitsNLC,
        simulatedRatio: simulatedRatio,
        indexingScheme: indexingScheme,
        indexAssignmentBitLength: indexAssignmentBitLength,
        indexed: resultsIndexed,
        spaceDefaults: spaceDefaults
    }

    // Return the results
    console.log("Dataset configured")
    console.log(results)
    return results
}

/** Analyzes the different possible indexing schemes to decide on the one to use for
implementation. The parameters were iterated from 15 to 1 for highestIndex, 1 to 3 for
indexAssignmentBitLength, and 2^indexAssignmentBitLength and 2^indexAssign-mentBitLength for
desiredIndexes. The results of the test are shown in Results of the indexing scheme analysis
in the Iteration 3 Write-Up. */
function analyseIndexingSchemes() {
    let test = ""
    let testingHighestIndex = 15
    let testingDesiredIndexes = 3
    let testingIndexAssignmentBitLength = 1

    for (let k = 15; k > 0; k--) {
        for (let j = 1; j <= 3; j++) {
            for (let i = 0; i < 2; i++) {
                let processedDataset = datasetConfig(k, Math.pow(2, j) - i, j, x16webRaw)

                test += `${k}\t${Math.pow(2, j) -
i}\t${j}\t${processedDataset.totalBitsASCII}\t${processedDataset.totalBitsNLC}\t${processedDat
aset.simulatedRatio}\t${processedDataset.indexingScheme}\n`
            }
        }
    }

    navigator.clipboard.writeText(test)
    alert("Copied to clipboard")
}

function preconfigCopy() {
    navigator.clipboard.writeText("let web15 = " + JSON.stringify(datasetConfig(15, 7, 3,
x16webRaw)))
}

function copyBitArray() {
    navigator.clipboard.writeText(encoded.default)
}
```

```
/** Returns binary of decimal. dec is decimal input. Length determines necessary leading
zeroes.
 * @param {number} dec Decimal Number
 * @param {number} length The minimum length of the binary output, adds leading zeroes if
necessary. Example: dec2bin(19, 8) → 00010011; dec2bin(19,0) → 10011
*/
function dec2bin(dec, length) {
    let bin = ""
    while (true) {
        bin = dec % 2 + bin
        dec = Math.floor(dec / 2)
        if (dec == 0) break
    }
    let binlen = bin.length
    for (let i = 0; i < length - binlen; i++) {
        bin = 0 + bin
    }
    return (bin)
}

/** Returns decimal of binary
 * @param {string} bin Binary number as a string
*/
function bin2dec(bin) {
    let binaryArray = bin.split("")
    let dec = 0
    for (let i = 0; i < binaryArray.length; i++) {
        dec += Math.pow(2, i) * binaryArray[binaryArray.length - i - 1]
    }
    return (dec)
}

/** Returns binary of a string encoded in ASCII
 * @param {string} input
*/
function ascii2bin(input) {
    // Credit: https://www.geeksforgeeks.org/javascript-program-to-convert-a-string-to-binary/
    let binaryResult = "";
    let binaryResultSpaced = "";

    for (let i = 0; i < input.length; i++) {
        const charCode = input.charCodeAt(i);
        let binaryValue = "";

        for (let j = 7; j >= 0; j--) {
            binaryValue += (charCode >> j) & 1;
        }

        binaryResultSpaced += binaryValue + " ";
        binaryResult += binaryValue;
    }

    return { default: binaryResult.trim(), spaced: binaryResultSpaced.trim() };
}

/*
// errorWord testing (deprecated)
function copyErrorWords() {
    testing = false
    console.log(errorWords)
    let letterFreq = [...freqLetters, XON, SUB]
    for (let i = 0; i < letterFreq.length; i++) {
        let letter = letterFreq[i]
        letterFreq[i] = [letter, errorWords.filter(x => x == letter).length]
    }
    let toCopy = totalWords + "\n"
    for (let i = 0; i < letterFreq.length; i++) {
        toCopy += letterFreq[i].join("\t") + "\n"
    }
    navigator.clipboard.writeText(toCopy)
}
*/

// Data Analysis
let testing = false
function commenceTesting() {
    testing = true
}

let testResults = []

/** When sentence testing is taking place, p5.js is used to continuously update frames after
each sentence */
function draw() {
    if (testing) {
        document.getElementById("og").value = sampleSentences[Math.floor(Math.random() *
sampleSentences.length)]//.toLowerCase()
        testResults.push(encodeInput())
        document.getElementById("test-stats").innerHTML = [
            `<tr><td>Sentences:</td><td>${testResults.length}</td></tr>`,
        ].join("\n")
    }
}

/** Copy the results of the sentence test to the clipboard */
function copyResults() {
    testing = false
    let toCopy = ""
    for (let i = 0; i < testResults.length; i++) {
        let x = testResults[i]
        toCopy += x.input + "\t" + x.bitsNLC + "\t" + x.bitsASCII + "\t" + x.ratio + "\n"
    }
    navigator.clipboard.writeText(toCopy)
}
```

./decode/index.html

```
<!DOCTYPE html>
<html>

<head>
  <script src="../web-15.js"></script>
  <title>Natural Language Compression</title>
  <link rel="stylesheet" href="../darkmode.css">
  <style>
    body {
      background-color: #181a1b;
      color: white
    }

    .input-container {
      padding-right: 7px;
      margin-bottom: 50px;
    }

    textarea {
      width: 100%;
      height: 100px;
      resize: vertical;
      font-size: medium;
    }

    p {
      font-family: monospace;
      font-size: medium;
      word-wrap: break-word;
      white-space: pre-wrap;
    }

    select {
      padding-bottom: 0px;
      padding-top: 0px;
    }

    #dataset-stats {
      margin-top: 0px;
      margin-bottom: 0px;
    }

    .indented {
      border-left-color: white;
      border-left-width: 1px;
      border-left-style: solid;
      padding-left: 8px;
    }

    h2 {
      font-size: 1.25em;
      margin-bottom: 0;
    }

    #stats {
      border-left-color: rgb(0, 255, 0);
    }

    a, a:visited {
```

```
    color: #3391ff;
  }

  button, input, textarea, select {
    background-color: #2b2a33;
    color: white;
    border-color: #8f8f9d;
    border-radius: 5px;
  }

  .tooltip {
    position: relative;
    display: inline-block;
  }

  .tooltip .tooltiptext {
    visibility: hidden;
    width: max-content;
    background-color: #2b2a33;
    color: #fff;
    text-align: left;
    padding: 5px 0;
    border-radius: 6px;

    position: absolute;
    z-index: 1;
  }

  .tooltip:hover .tooltiptext {
    visibility: visible;
  }

  .tooltip .tooltiptext {
    bottom: 100%;
    left: 50%;
    margin-left: -50%;
  }

  .flex-container {
    display: flex;
    justify-content: space-between;
  }

  #testing-menu {
    border-right-color: white;
    border-right-width: 1px;
    border-right-style: solid;
    padding-right: 8px;
    display: flex;
    flex-direction: column;
    align-items: end;
  }

  #testing-menu * {
    margin-top: 4px;
  }

  .dataAnalysis {
    visibility: hidden;
  }
</style>
```

```
</head>

<body>
  <main>
    <a href="./..">Natural Language Encoder</a>
    <h1>Natural Language Decoder v3</h1>
    <p id="stats"></p>
    <div class="dropdown-container">
      <label for="wordModel-select">NLC Word Model:</label>
      <select id="wordModel-select" name="wordModel-select" onchange="changeWordModel()">
        <option value="web333k">333k-web</option>
        <option value="wikipedia1k">1k-wikipedia</option>
      </select>
    </div><br>
    <div class="input-container">
      <label for="og">NLC Binary Input</label>
      <button onclick="decodeInput()">Decode</button>
      <textarea id="og" name="og">1101000100001011111011111010100</textarea>
    </div>
    <div class="output-container">
      <h2 for="outin">Output</h2>
      <p type="text" id="outin" name="outin"></p>
    </div>
    <script src="./script.js"></script>
  </main>
</body>
</html>
```

./decode/script.js

```
const XON = "␑" // Space override (ASCII Device Control One (XON)) is set as a constant,
because ␑ is difficult to type and not monospaced
const DC2 = "␒" // No space override (ASCII Device Control One (XON)) is set as a constant,
because ␑ is difficult to type and not monospaced
const SUB = "�" // Unknown substitute character to substitute any unencodable lexemes (not
monospaced so it's a constant)

function decodeInput() {
    document.getElementById("outin").innerHTML = decode(document.getElementById("og").value,
web15)
}

function decode(encoded, dataset) {
    //Seperate index assignments until the sum of their indexes and the index is the length of
encoded
    let indexArray = []
    let i = 0
    let j = dataset.indexAssignmentBitLength
    while (sum(...indexArray, i) < encoded.length) {
        indexArray.push(dataset.indexingScheme[bin2dec(encoded.substring(i, j))])
        i += dataset.indexAssignmentBitLength
        j += dataset.indexAssignmentBitLength
    }

    // Get lexemes
    let lexemeArray = []
    j = i
    for (let k = 0; k < indexArray.length; k++) {
        j += indexArray[k]
        lexemeArray.push(web15.indexed.lexemes[(bin2dec(encoded.substring(i, j)))])
        i += indexArray[k]
    }

    // Spacing
    let output = ""
    for (let i = 0; i < lexemeArray.length; i++) {
        const pre = lexemeArray[i - 1] // Previous
        const cur = lexemeArray[i] // Current
        const nex = lexemeArray[i + 1] // Next
        function ns(lex) { return dataset.spaceDefaults.noSpaces.includes(lex) } // No space
        function rs(lex) { return dataset.spaceDefaults.rightSpace.includes(lex) } // Right
space
        function ls(lex) { return dataset.spaceDefaults.leftSpace.includes(lex) } // Left
space

        if (ns(cur) || rs(cur)) {
            output += cur
        } else if (ns(pre)) {
            output.slice(0, -1)
            output += cur
        } else if (!ls(pre)) {
            output += " " + cur
        } else {
            output += cur
        }
    }


    return output.trim().replaceAll(XON, " ").replaceAll(DC2, "")
}

function sum() {
    let sum = 0
    for (let i = 0; i < arguments.length; i++) {
        sum += arguments[i]
    }
    return sum
}

function dec2bin(dec, length) {
    // returns binary of decimal. dec is decimal input. length determines necessary leading
zeroes
    // ex. dec2bin(19, 8) => 00010011 dec2bin(19,0) => 10011
    let bin = ""
    while (true) {
        bin = dec % 2 + bin
        dec = Math.floor(dec / 2)
        if (dec == 0) break
    }
    let binlen = bin.length
    for (let i = 0; i < length - binlen; i++) {
        bin = 0 + bin
    }
    return (bin)
}

function bin2dec(bin) {
    let binaryArray = bin.split("")
    let dec = 0
    for (let i = 0; i < binaryArray.length; i++) {
        dec += Math.pow(2, i) * binaryArray[binaryArray.length - i - 1]
    }
    return (dec)
}


function ascii2bin(input) {
    // Credit: https://www.geeksforgeeks.org/javascript-program-to-convert-a-string-to-binary/
    let binaryResult = "";
    let binaryResultSpaced = "";

    for (let i = 0; i < input.length; i++) {
        const charCode = input.charCodeAt(i);
        let binaryValue = "";

        for (let j = 7; j >= 0; j--) {
            binaryValue += (charCode >> j) & 1;
        }

        binaryResultSpaced += binaryValue + " ";
        binaryResult += binaryValue;
    }

    return { default: binaryResult.trim(), spaced: binaryResultSpaced.trim() };
}
```