INTERACTION WITH LARGE SCALE MODELS

LIACD/1

University of Beira Interior,
Department of Informatics

Hugo Pedro Proença,
hugomcp@di.ubi.pt, 2024/2025

INTERACTION WITH LARGE SCALE MODELS

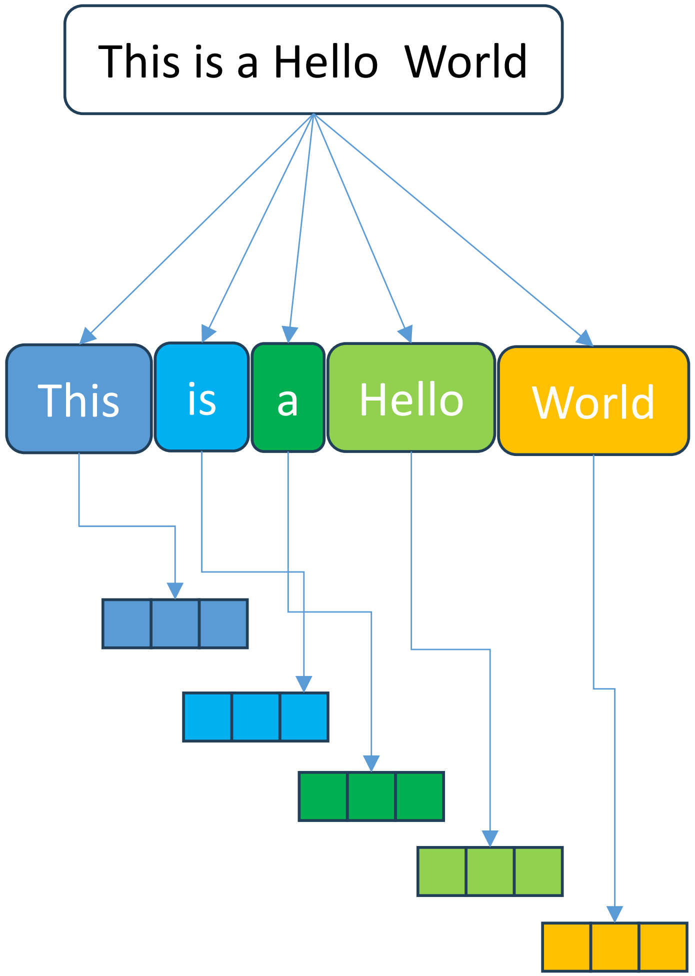
[02]

Syllabus

- Tokenization: How text is processed into tokens
- Embeddings: How models understand word meaning

Tokenization

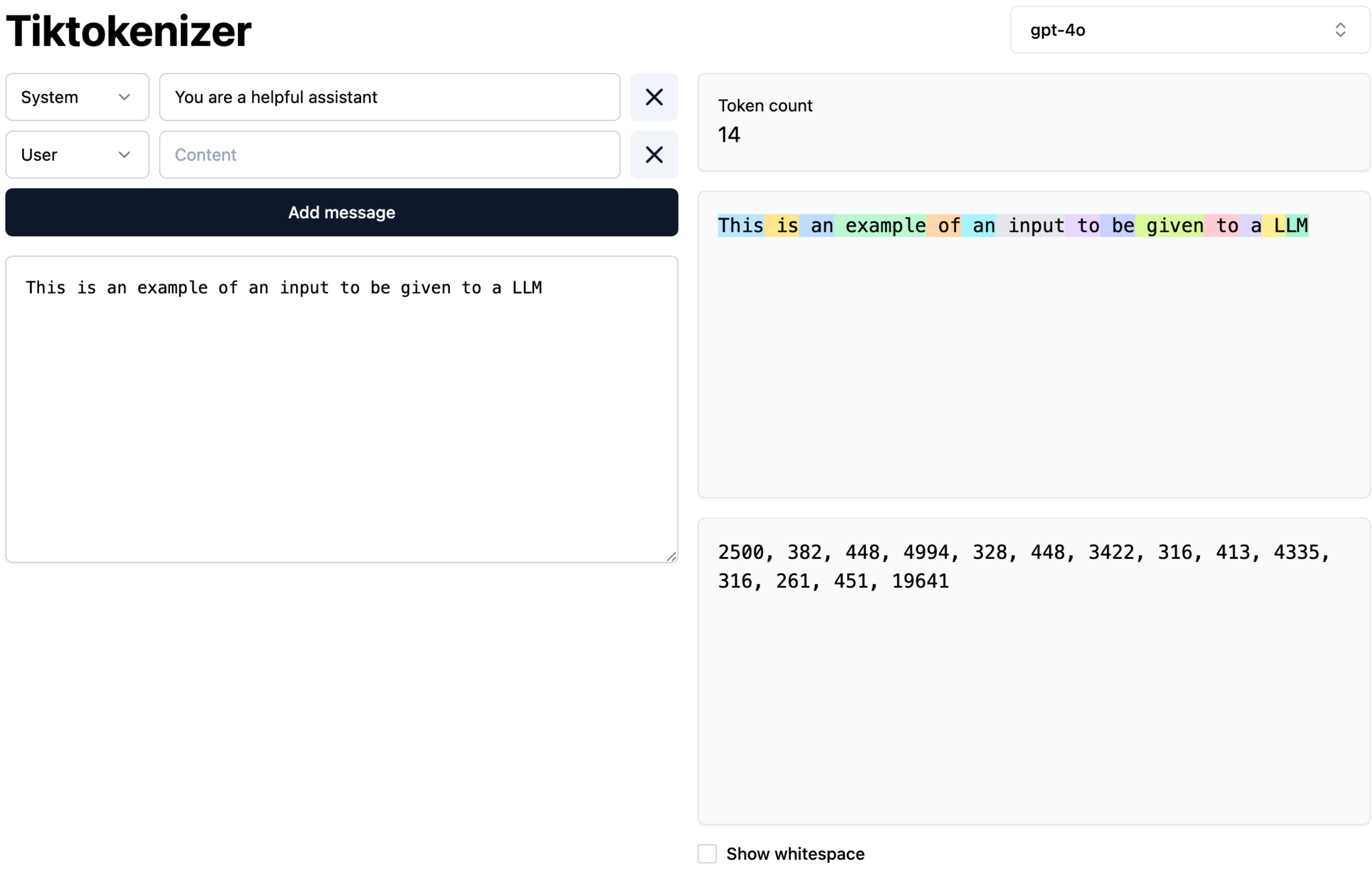
- **Tokens** are the fundamental unit, the “atom” of Large Language Models (LLMs). Tokenization is the process of translating strings and converting them into sequences of tokens.
- In practice, it regards breaking the input into smaller units (tokens) that models can process
    - **Word-based**: Splitting by words (e.g., "Prompt Engineering" → ["Prompt", "Engineering"])
    - **Subword-based** (Byte-Pair Encoding - BPE): Splitting into frequent character sequences (e.g., "Engineering" → ["Eng", "ineering"])
    - **Character-based:** Splitting into individual characters (e.g., "Engineering" → ["E", "n", "g", "i", "n", "e", "e", "r", "i", "n", "g"])



Tokenization has evident impact in input size and model efficiency

Also, it determines context length and processing speed

Tokenization - Example

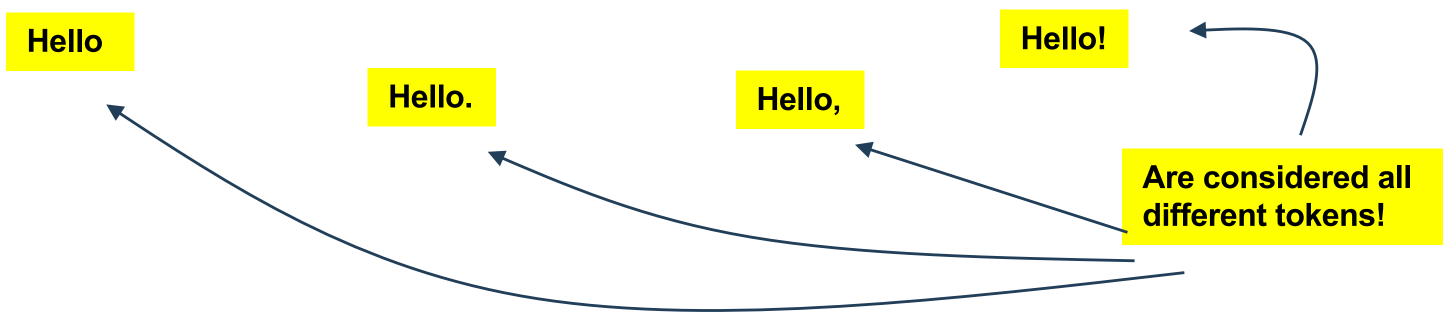


Compare the tokens used for different LLMs

Source: https://tiktokenizer.vercel.app/

Tokenization

- Most of the recent SOTA models make use of — Byte-Pair Encoding (BPE), Word Piece, Unigram, and Sentence Piece.
- Word Based Tokenization
  - As the name suggests, in word-based tokenization methods entire words separated by either punctuation, whitespaces, delimiters, etc. are considered as tokens.
  - The simplest approach might consider whitespaces as delimiters only.
  - However, using it on a bigger dataset this method would result in a huge vocabulary as innumerable groupings of words and punctuations would be treated as different tokens from the underlying word.



Tokenization

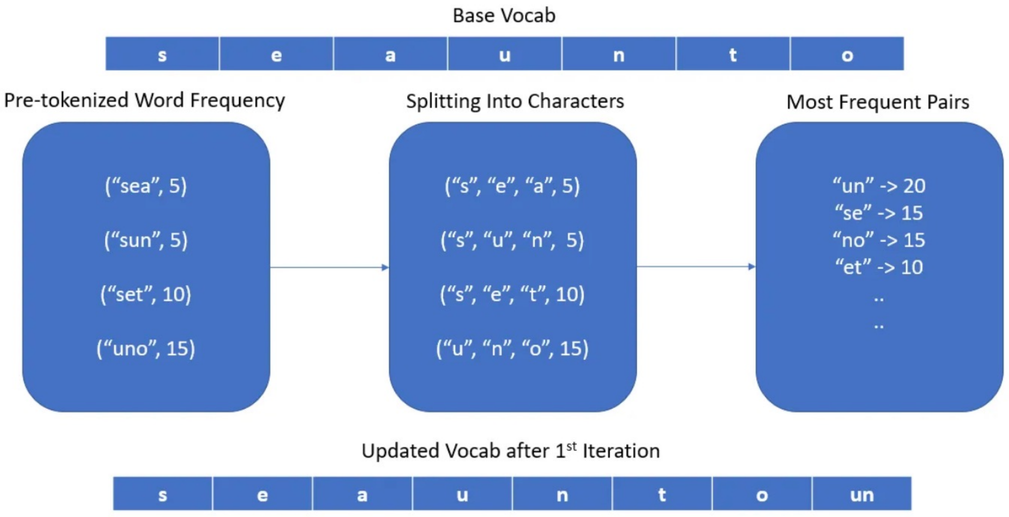
- Rule-based words tokenizers.
  - Consider the semantics of each specific language, to define rules for breaking words into tokens
- SpaCy offers a great rule-based tokenizer which applies rules specific to a language for generating semantically rich tokens. Interested readers can take a sneak peek into the rules defined by spacy.
- Character Based Tokenization
  - Has the main advantage of reducing drastically the complexity and size of the vocabulary (up to almost 200 tokens)
  - However, the tokens no longer carry meaningful semantics. To put this into context, representations of “king” and “queen” in word-based tokenization would contain a lot more information than the contextual-independent embeddings of letter “k” and “q”. This is the reason why language models perform poorly when trained on character based tokens.

Tokenization

- **Sub-word Based Tokenization**
  - Aim to represent all the words in dataset by only using as many as N tokens, where N is a hyperparameter and is decided as per your requirements. Generally, for base models, it hovers around ~30,000 tokens. Thanks to these methods, without needing an infinite vocabulary we're now well equipped to capture context-independent semantically rich token representations.
- There are four main algorithms used for this type of tokinzation
  - Byte-Pair Encoding (BPE)
  - Word Piece
  - Unigram
  - Sentence Piece

Byte-Pair Encoding

- It requires the input to be pre-tokenized — which can be a simple whitespace tokenization or a rule-based tokenizer such as SpaCy.
- Next, a base vocabulary is created which is a collection of all unique characters in the corpus. We also obtain the frequency of each token and represent each token as a list of individual characters from base vocabulary.
- Then, merging begins. Tokens are added to our base vocab if the maximum size is not reached
  - The pair of tokens occurring the most times is merged and considered as a new token. This step is repeated until we reach the configured maximum vocab size.



Source: https://medium.com/towards-data-science/tokenization-algorithms-explained-e25d5f4322ac

Word Piece

- Word Piece and Byte-Pair Encoding (BPE) are very similar in their approaches of achieving sub-word tokenization.
- The BPE's primary criterium is to select the candidate pair with the maximum frequency.
- Word Piece, instead, focuses on maximizing the likelihood of a candidate pair (ab), with "a" and "b" being tokens.
- Intuitively, WordPiece is slightly different to BPE in that it evaluates what it loses by merging two symbols to ensure it's worth it.

$$\frac{P(ab)}{P(a)\,P(b)}$$

Sentence Piece

- Surprisingly, it's not actually a tokenizer
- It's actually a method for selecting tokens from a precompiled list, optimizing the tokenization process based on a supplied corpus.
- We start by assessing the probability P(Y|X) of a target sentence Y given an input sentence X.

$$P(Y|X;\theta) = \prod_{i=1}^{n} P(y_i|\mathbf{x}, y_{<i};\theta)$$

- Note that X and Y can be formed by a very large number of sub-word sequences.
  - For instance, 'translation' can be tokenized in many different ways (e.g., [t, r,an, s, l, a ti, o n] or [t, r, a, ns, la, ti, o n], …
- This algorithm not only considers this fact, but it takes advantage from it.

Sentence Piece

- The cost function is given by:

$$\mathcal{L}(\theta) = \sum_{s=1}^{|D|} \mathbb{E}_{\substack{x\sim P(\mathbf{x}|X^{(s)})\\ y\sim P(\mathbf{y}|Y^{(s)})}} \log P(\mathbf{y}|\mathbf{x};\theta)$$

  - Where |D| is the number of possible tokenizations (segmentations), and “x” and “y” and drawn from their distributions over all possible segmentations.
- In practice, we remove the expectation E[…] and just use “x” and “y” as a single randomized segmentation.
- However, the main problem is to build a probability distribution over an exponential number of states.
  - As in most optimization problems, we obtain an approximate solution.
  - Assuming that it is too hard to obtain joint distributions P(x_1, x_2), we assume that:

$$p(\mathbf{x}|X) \approx \prod_{i=1}^{n} p(x_i) \qquad \sum_{x\in V} p(x) = 1$$

Sentence Piece

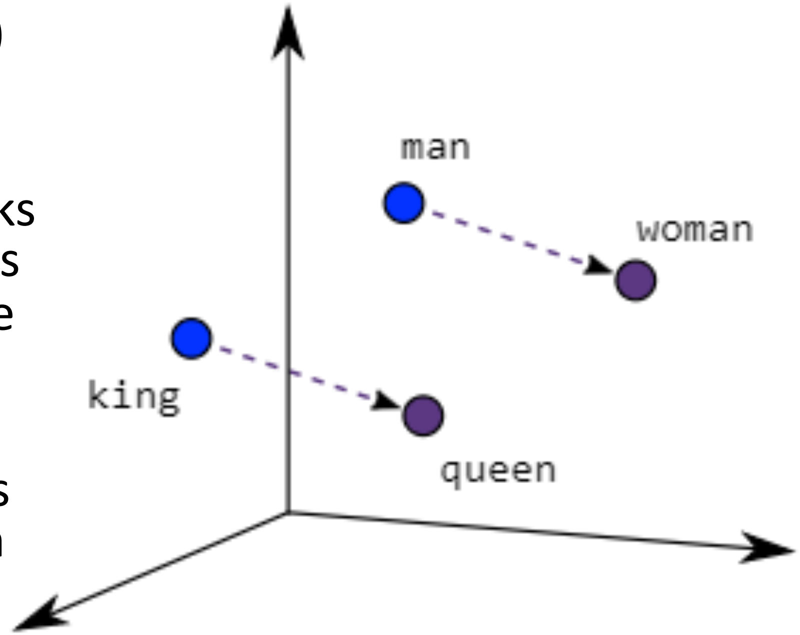
- To train, we want to maximize the log-probability of obtaining a particular tokenization _X=(x_1, …, x_n) of the corpus, given the unigram probabilities $p(x_1)$,…,$p(x_n)$. Note that only the full un-tokenized sequence X is observed.
- The overall training objective is given by:

$$\mathcal{L} = \sum_{s=1}^{|D|} \log P(X^{(s)}) = \sum_{s=1}^{|D|} \log \left(\sum_{\mathbf{x} \in S(\mathbf{x})} P(\mathbf{x}) \right)$$

  - Where "x" are the unigram sequences, and S(x) denotes the set of all possible sequences.
- To solve this, we incorporate an Expectation-Maximization (EM) type algorithm.
  - Initialize the unigram probabilities.
  - REPEAT
    - **M-step:** compute the most probable unigram sequence given the current probabilities.
      - If all of the sub-words were of the same length, this would be a classic application of the Viterbi algorithm. Instead, it is solved with dynamic programming.
    - **E-step:** given the current tokenization, recompute the unigram probabilities by counting the occurrence of all sub-words in the tokenization.

Embeddings

- Token feature embeddings are crucial in foundational models because they transform discrete tokens (words, sub-words, or characters) into dense vector representations that capture semantic, syntactic, and contextual information.
- Unlike one-hot encoding, which is sparse and lacks meaningful relationships, embeddings allow LLMs to understand similarities between words, handle polysemy (words with multiple meanings), and generalize across different linguistic patterns.
- These embeddings also enable models to process unseen words by leveraging similarities to known ones, improving robustness and adaptability.
- Additionally, specialized embeddings, such as position and segment embeddings, help models encode word order and structural relationships, further enhancing their ability to generate coherent and contextually relevant text.



Source: https://www.naukri.com/code360/library/word2vec

Embeddings

- As a (non-real) illustration, imagine a set of independent and human-interpretable features, where the corresponding coefficient for each word will represent *how much* the word relates to that feature (property).
  - *In practice, features are not independent and interpretable.*

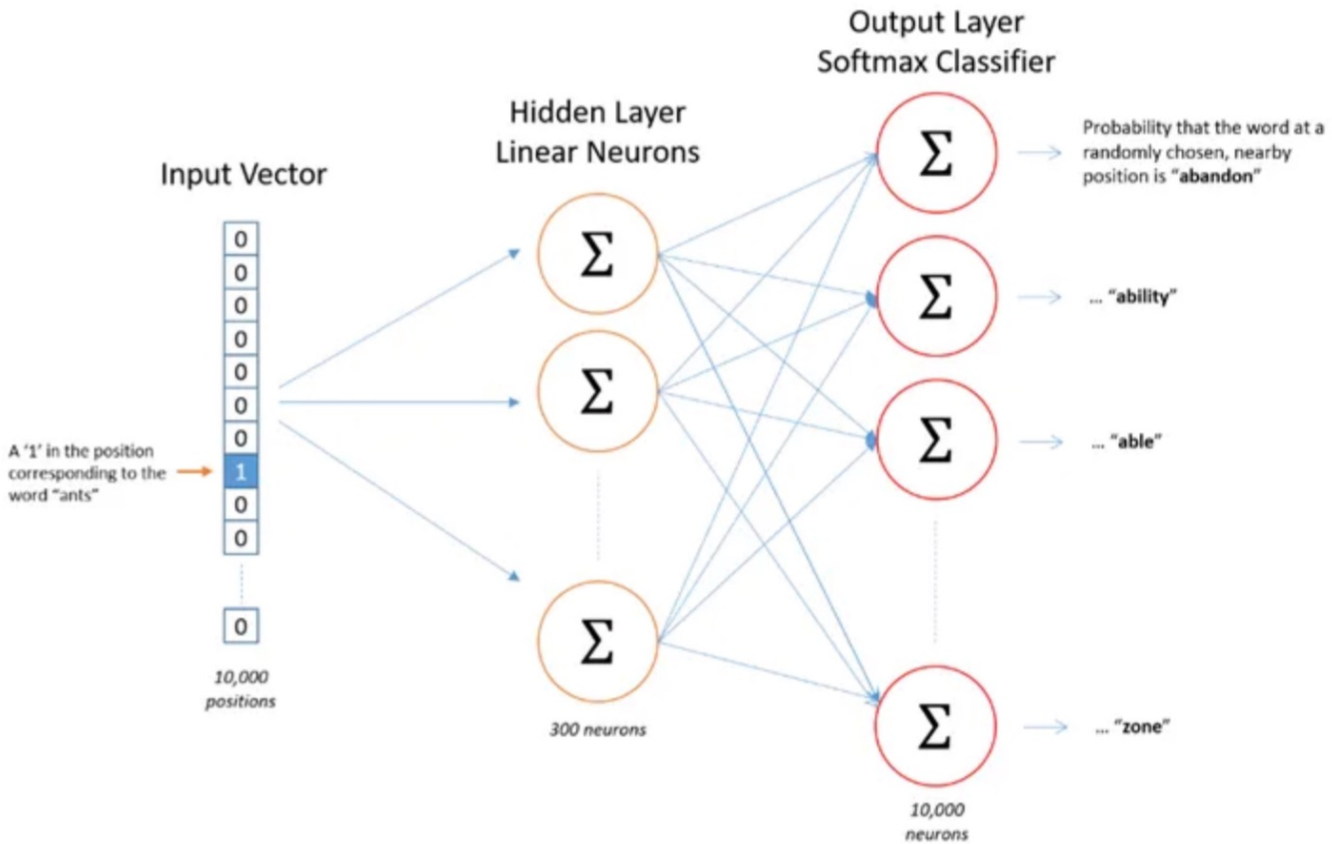
Source: https://medium.com/@manansuri/a-dummys-guide-to-word2vec-456444f3c673

Embeddings

- *Word2Vec* is among the most popular methods for obtaining semantically consistent language embeddings.
- The main idea is that, instead of representing words as one-hot encoding in high dimensional space, to represent words in a dense low dimensional space, in a way that similar words get similar word vectors, so they are mapped to nearby points.
- At the top scale, the algorithm is given by:
  - Move through the training corpus with a sliding window: Each word is a prediction problem.
  - The objective is to predict the current word using the neighbor words (or vice versa).
  - The outcome of the prediction determines whether we adjust the current word vector. Gradually, vectors converge to (hopefully) optimal values.
- The prediction itself is not the final goal. It is a proxy to learn vector representations that can be used in other tasks.

Embeddings

- One of the simplest architectures is just a simple one hidden layer plus one output layer.



Source: https://medium.com/towards-data-science/understanding-word2vec-embedding-in-practice-3e9b8985953

Embeddings

- During training, we should maximize the probability of predicting a word ω_t given its surrounding words (backward and forward).

$$P(\omega_t \mid \omega_{t-n},\dots, \omega_{t-1}, \omega_{t+1}, \dots, \omega_{t+n})$$

- This is approximated by the number of times the word is observed in that context, divided by the number of times the context is observed (with or without the word).
- Thus, the final loss formulation is given by the negative log-likelihood:

$$\mathrm{J} = -\sum log\, P(\omega_t \mid \omega_{t-n},\dots, \omega_{t-1}, \omega_{t+1}, \dots, \omega_{t+n})$$

- This way, it learns word embeddings by optimizing a neural network that predicts word-context relationships.