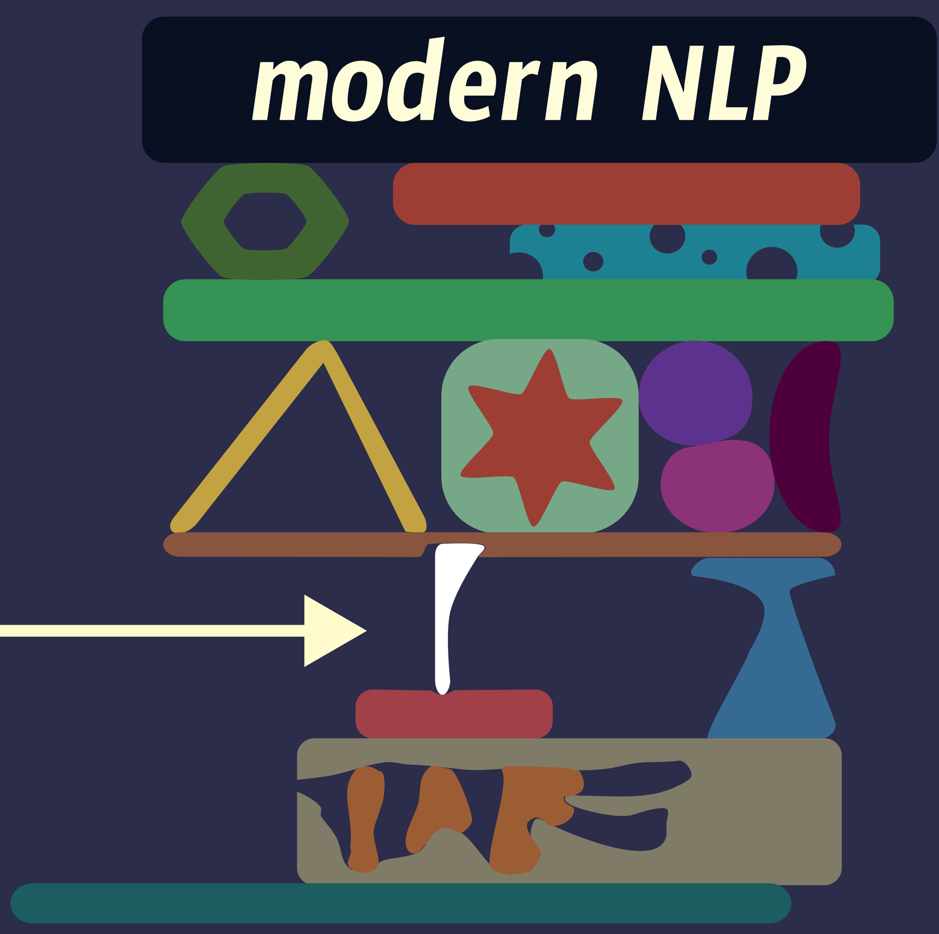
What even is Byte-Pair Encoding?



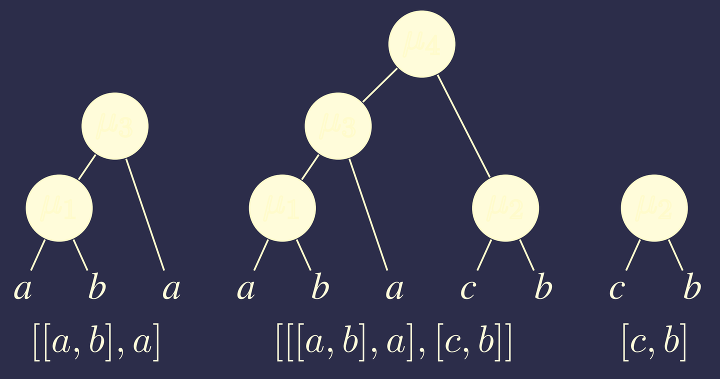
rapacious zealous aardvarks $\xrightarrow{\text{BPE}}$ *rapac ··i ··ous zeal ··ous aardvark ··s*

```python
from collections import Counter
from typing import Union, Tuple, List

def bpe(xs: Union[str, List], V: int):
  for _ in range(V):
    pairs = Counter(zip(xs, xs[1:]))
    top_pair = pairs.most_common(1)[0
    xs = merge(list(xs), top_pair)
  return xs

def merge(xs: List, pair: Tuple):
  ys = []
  while xs:
    if tuple(xs[:2]) == pair:
      ys.append(pair)
      xs = xs[2:]
    else:
      ys.append(xs.pop(0))
  return ys
```

enough to make your NLP application work
not enough to understand it



$x = \langle A\ A\ B\ B\ A\ A\ C\ C\ A\ A\ A\ A\ C\ A\ A\ A\ A\ D \rangle$

$\mu_1 = \langle [A\ A], [B\ B], [\ AA,\ AA]\ \rangle$
$\text{Apply}(\mu_1, x) = \langle [AA]\ [BB]\ [AA]\ CC\ [AAAA]\ C\ [AAAA]\ D \rangle$

$\mu_2 = \langle [A\ C], [B\ B], [C\ A] \rangle$
$\text{Apply}(\mu_2, x) = \langle A\ A\ [BB]\ A\ [AC]\ [CA]\ A\ A\ [AC]\ A\ A\ A\ A\ D \rangle$

Algorithm Iterative Greedy BPE
Inputs: sequence $\boldsymbol{x}$, merge count M
Output: merge sequence $\boldsymbol{\mu}$, tokenized sequence $\boldsymbol{x}$

1: $\boldsymbol{\mu} \leftarrow \langle \rangle$
2: **for** i in $\{0, \ldots, M\}$ **do**
3: $\quad \mu \leftarrow \underset{(\mu', \mu'') \in \text{set}(\boldsymbol{x})^2}{\text{argmax}} \ \text{PAIRFREQ}(\boldsymbol{x}, (\mu', \mu''))$
4: $\quad \boldsymbol{x} \leftarrow \text{APPLY}(\mu, \boldsymbol{x})$
5: $\quad \boldsymbol{\mu} \leftarrow \boldsymbol{\mu} \circ \langle \mu \rangle$
6: **end for**
7: **return** $\boldsymbol{\mu}, \boldsymbol{x}$

Why is μ_1 better than μ_2 ?
$\kappa(\mu_1) = |x| - |\text{Apply}(\mu_1)| = 8 > 4 = |x| - |\text{Apply}(\mu_2)| = \kappa(\mu_2)$

What's the optimal $\mu^* = \text{argmax}\ \kappa(\mu)$?
How does it relate to $\mu^\dagger = \text{GreedyBPE}(x)$?

$$\kappa(\mu^\dagger) \ / \ \kappa(\mu^*) \geq 0.37$$

Greedy BPE will compress at worst 3 times less effectively than the optimum

How do we know? The utility function κ is "sequence submodular" and has some other nice properties that lead to the approximation bound.

In the paper:
- *Proper formalization*
- *Runtime analysis of naive & faster implementations O(N M) & O(N log M)*
- *Algorithm for optimal BPE merge sequence*

```
p i c k e d      p i c k l e d      p i c k l e s
pi c k e d       pi c k l e d       pi c k l e s
pi ck e d        pi ck l e d        pi ck l e s
pick e d         pick l e d         pick l e s
pick ed          pick l ed          pick l e s
pick ed          pickl ed           pickl e s
```

Greedy
[a,b]a[a,b]baa ab: 2, ba: 2, aa: 2, bb: 1
[[a,b],a][a,b]baa [a,b]a: 1, [a,b]b: 1, ba: 1, aa:1, [a,[a,b]]: 1
Optimal
a[b,a]ab[b,a]a ab: 2, ba: 2, aa: 2, bb: 1
a[[b,a],a]b[[b,a],a] ab: 2, a[b,a]: 1, [b,a]a: 2, b[b,a]: 1

vzouhar@ethz.ch
Vilém Zouhar, Clara Meister, Juan Luis Gastaldi, Li du, Tim Vieira, Mrinmaya Sachan, Ryan Cotterell