COMP 3361 Natural Language Processing

Lecture 8: Neural language models: Tokenization

Spring 2025

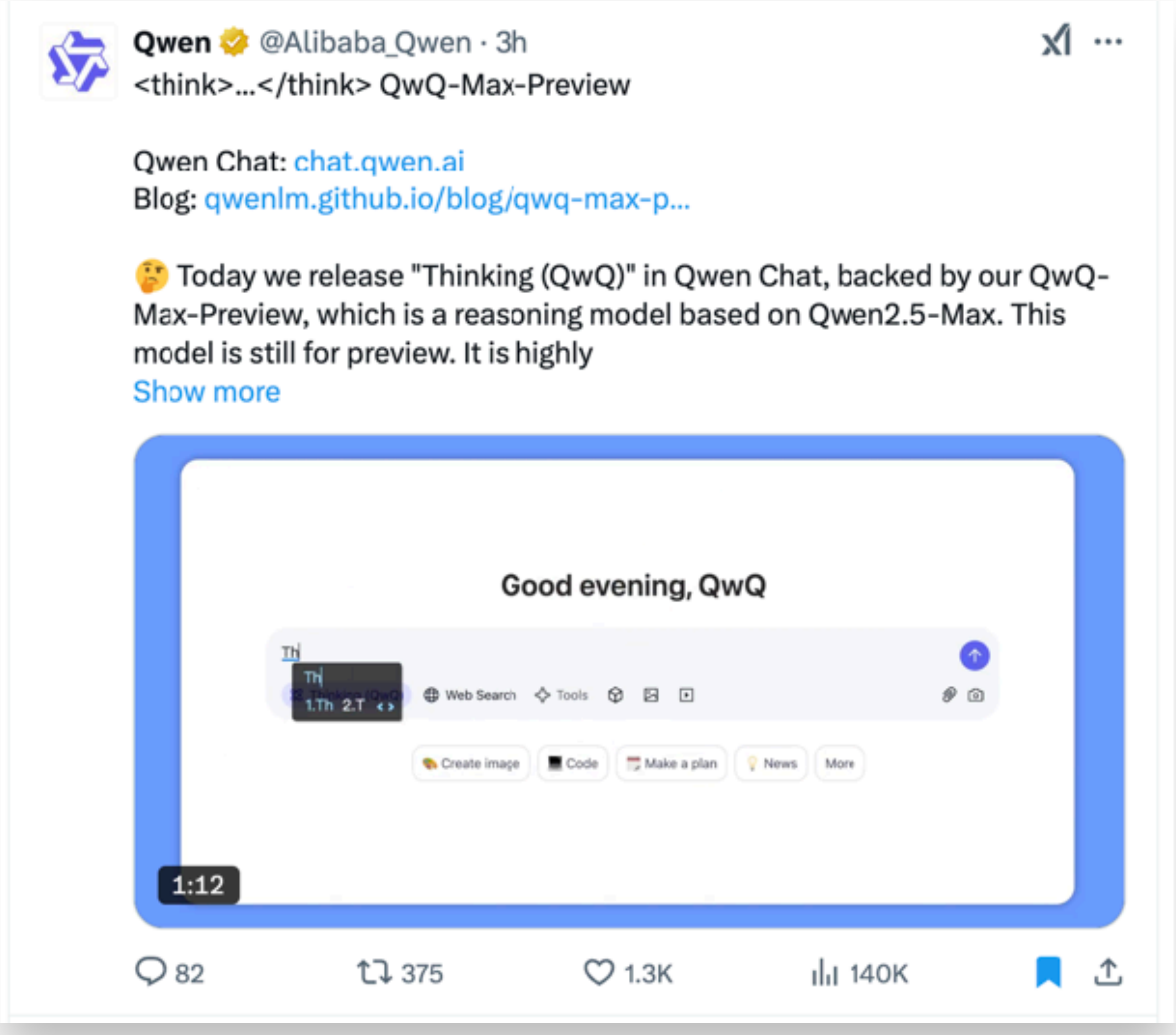
Many materials from CSE447@UW (Taylor Sorensen) and COS484@Princeton with special thanks!

Announcements

- Assignment 1 is due on Mar 4!
  - Will provide a short coding tutorial next Friday
  - Book a TA slot via the link on the course page
  - Also you can always ask questions on Slack

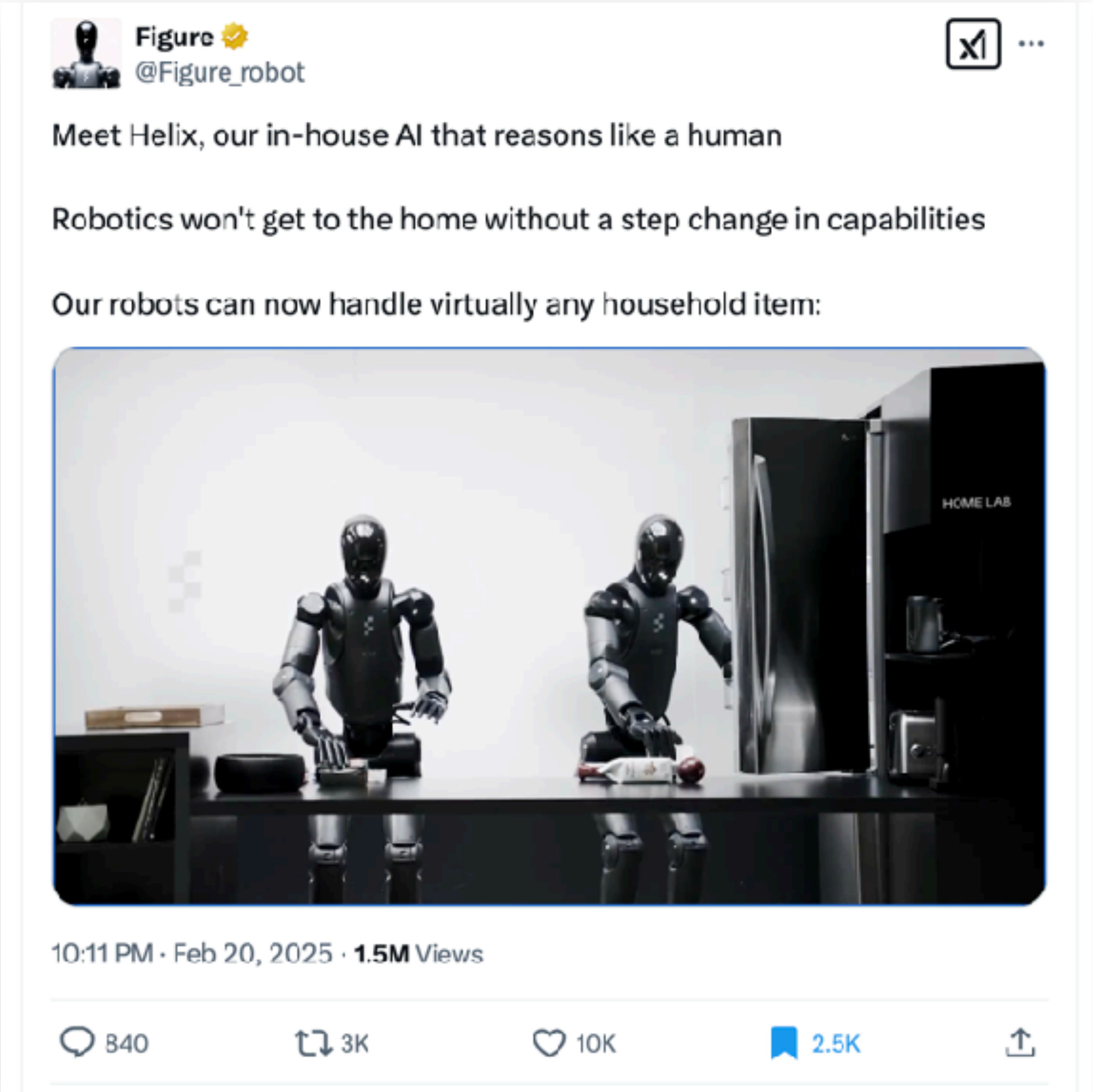
Latest AI news

- Try Grok 3 for free (access w/o VPN in HK): https://x.com/i/grok
- OpenAI roadmap update for GPT-4.5 and GPT-5 (coming in May)
- Anthropic Claude 3.7 Sonnet is out, also QwQ-Max-Preview



Latest AI news

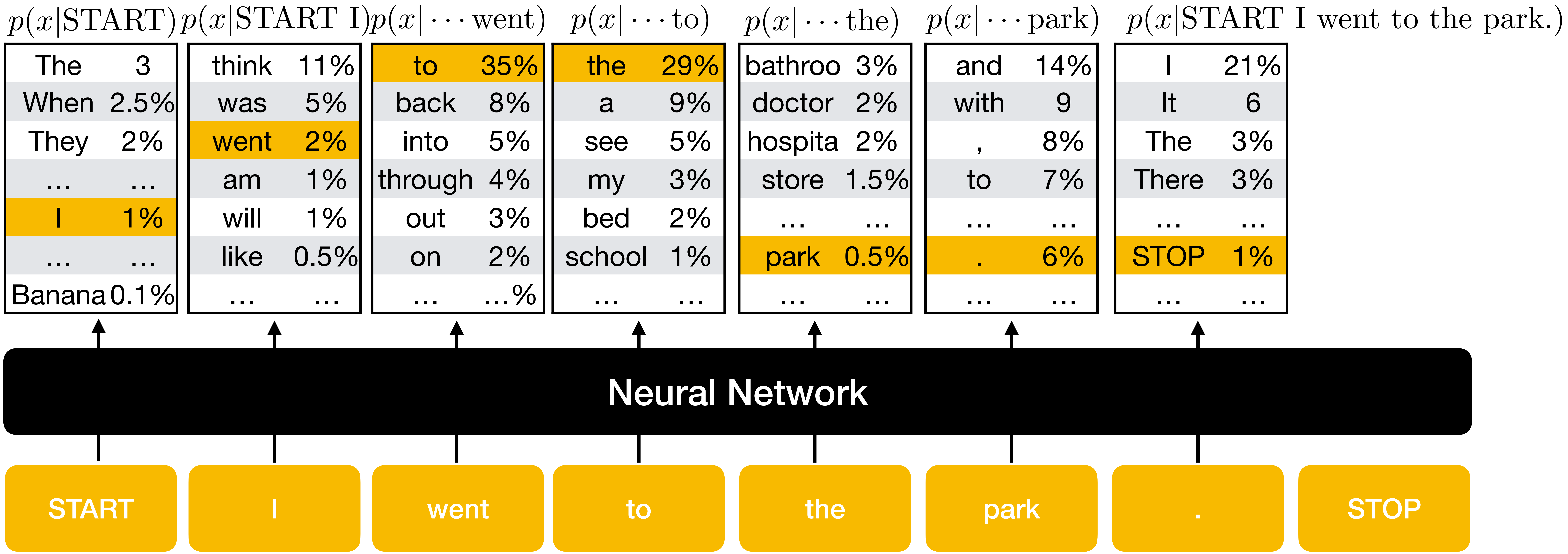
- Helix: A Vision-Language-Action Model for Generalist Humanoid Control



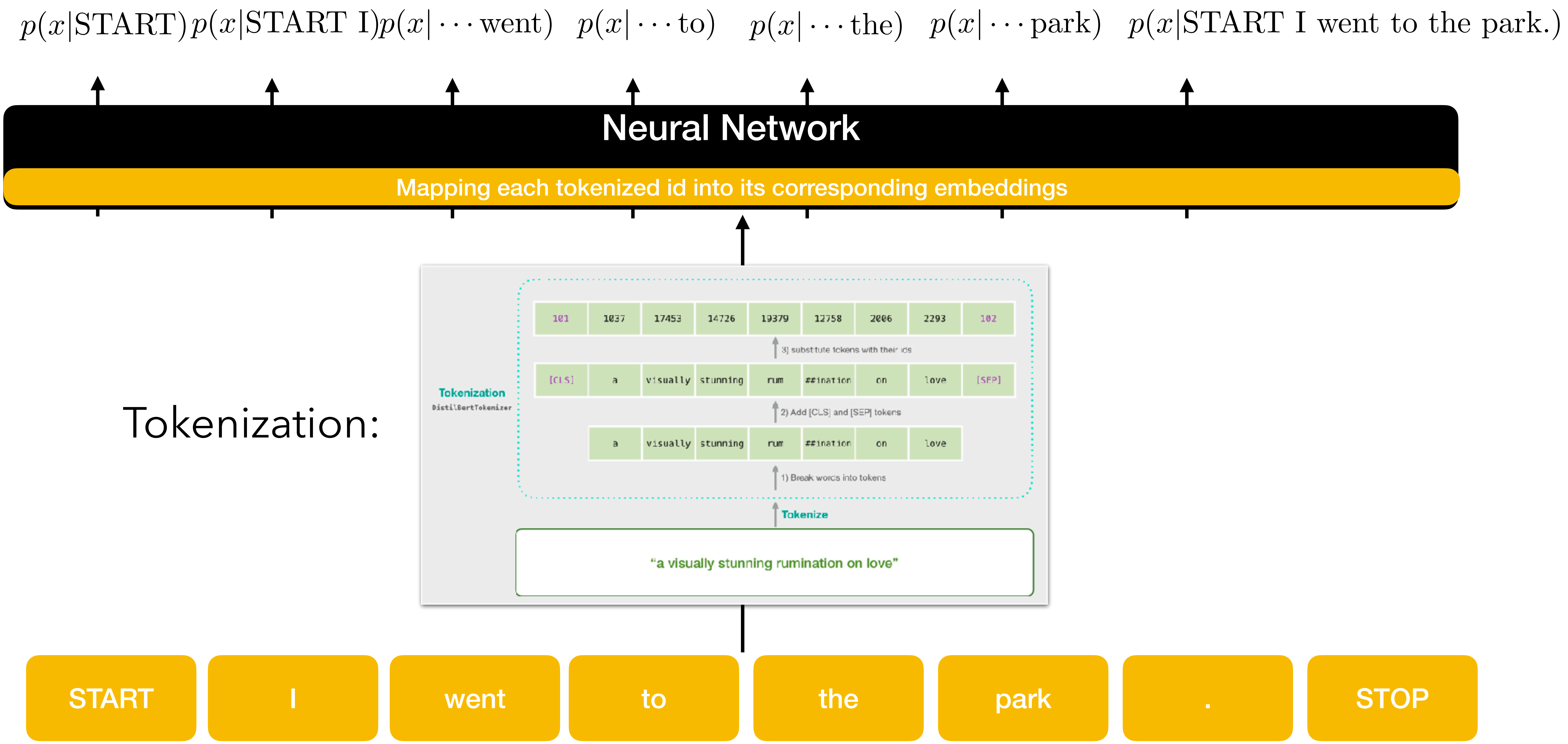
Neural language models: tokenization

Neural language models: inputs/outputs

- **Input:** sequences of words (or tokens)
- **Output:** probability distribution over the next word (token)



Tokenization to input vectors



ChatGPT tokenization example

Call me Ishmael. Some years ago—never mind how long precisely—having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral principle to prevent me from deliberately stepping into the street, and methodically knocking people's hats off—then, I account it high time tozz get to sea as soon as I can. This is my substitute for pistol and ball. With a philosophical flourish Cato throws himself upon his sword; I quietly take to the ship. There is nothing surprising in this. If they but knew it, almost all men in their degree, some time or other, cherish very nearly the same feelings towards the ocean with me.

TEXT TOKEN IDS

Tokens 239

Characters 1109

[7368, 757, 57704, 1764, 301, 13, 4427, 1667, 4227, 2345, 37593, 4059, 1268, 1317, 24559, 2345, 69666, 2697, 477, 912, 3300, 304, 856, 53101, 11, 323, 4400, 4040, 311, 2802, 757, 389, 31284, 11, 358, 3463, 358, 1053, 30503, 922, 264, 2697, 323, 1518, 279, 30125, 727, 961, 315, 279, 1917, 13, 1102, 374, 264, 1648, 358, 617, 315, 10043, 1022, 279, 87450, 268, 323, 58499, 279, 35855, 13, 43633, 358, 1505, 7182, 7982, 44517, 922, 279, 11013, 26, 15716, 433, 374, 264, 41369, 11, 1377, 73825, 6841, 304, 856, 13836, 26, 15716, 358, 1505, 7182, 4457, 3935, 6751, 7251, 985, 1603, 78766, 83273, 11, 323, 12967, 709, 279, 14981, 315, 1475, 32079, 358, 3449, 26, 323, 5423, 15716, 856, 6409, 981, 636, 1778, 459, 8582, 1450, 315, 757, 11, 430, 433, 7612, 264, 3831, 16033, 17966, 311, 5471, 757, 505, 36192, 36567, 1139, 279, 8761, 11, 323, 1749, 2740, 50244, 1274, 753, 45526, 1022, 2345, 3473, 11, 358, 2759, 433, 1579, 892, 311, 10616, 636, 311, 9581, 439, 5246, 439, 358, 649, 13, 1115, 374, 856, 28779, 369, 40536, 323, 5041, 13, 3161, 264, 41903, 67784, 356, 4428, 3872, 5678, 5304, 813, 20827, 26, 358, 30666, 1935, 311, 279, 8448, 13, 2684, 374, 4400, 15206, 304, 420, 13, 1442, 814, 719, 7020, 433, 11, 4661, 682, 3026, 304, 872, 8547, 11, 1063, 892, 477, 1023, 11, 87785, [illegible], [illegible], [illegible], 1890, 16024, 7119, 279, 18435, 449, 757, 13]

TEXT **TOKEN IDS**

https://platform.openai.com/tokenizer

Vocabulary: word-level

- For the n-gram model, our vocabulary $\mathcal{V}$ was comprised of all of the words in a language
- Some problems with this:
  - $|\mathcal{V}|$ **can be quite large** - ~470,000 words Webster's English Dictionary (3rd edition)
  - **Language is changing all of the time** - 690 words were added to Merriam Webster's in September 2023 ("rizz", "goated", "mid")
  - **Long tail of infrequent words**. Many words just occur a few times
  - **Some words may not appear** in a training set of documents
  - **No modeled relationship between words** - e.g., "run", "ran", "runs", "runner" are all separate entries despite being linked in meaning

Character-level?

What about representing text with characters?

- $V = \{a, b, c, \ldots, z\}$
  - (Maybe add capital letters, punctuation, spaces, …)
- Pros:
  - Small vocabulary size ($|V| = 26$ for English)
  - Complete coverage (unseen words are represented by letters)
- Cons:
  - Encoding becomes very long - # chars instead of # words
  - Poor inductive bias for learning

~~Word~~ ~~Character~~ <u>Subword</u> tokenization!

How can we combine the high coverage of character-level representation with the efficiency of word-level representation?

Subword tokenization! (e.g., Byte-Pair Encoding)

- Start with character-level representations
- Build up representations from there

Original BPE Paper (Sennrich et al., 2016)

https://arxiv.org/abs/1508.07909

Byte-pair encoding: ChatGPT example

Call me Ishmael. Some years ago—never mind how long precisely—having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral principle to prevent me from deliberately stepping into the street, and methodically knocking people's hats off—then, I account it high time tozz get to sea as soon as I can. This is my substitute for pistol and ball. With a philosophical flourish Cato throws himself upon his sword; I quietly take to the ship. There is nothing surprising in this. If they but knew it, almost all men in their degree, some time or other, cherish very nearly the same feelings towards the ocean with me.

TEXT TOKEN IDS

Tokens 239

Characters 1109

[7368, 757, 57704, 1764, 301, 13, 4427, 1667, 4227, 2345, 37593, 4059, 1268, 1317, 24559, 2345, 69666, 2697, 477, 912, 3300, 304, 856, 53101, 11, 323, 4400, 4040, 311, 2802, 757, 389, 31284, 11, 358, 3463, 358, 1053, 30503, 922, 264, 2697, 323, 1518, 279, 30125, 727, 961, 315, 279, 1917, 13, 1102, 374, 264, 1648, 358, 617, 315, 10043, 1022, 279, 87450, 268, 323, 58499, 279, 35855, 13, 43633, 358, 1505, 7182, 7982, 44517, 922, 279, 11013, 26, 15716, 433, 374, 264, 41369, 11, 1377, 73825, 6841, 304, 856, 13836, 26, 15716, 358, 1505, 7182, 4457, 3935, 6751, 7251, 985, 1603, 78766, 83273, 11, 323, 12967, 709, 279, 14981, 315, 1475, 32079, 358, 3449, 26, 323, 5423, 15716, 856, 6409, 981, 636, 1778, 459, 8582, 1450, 315, 757, 11, 430, 433, 7612, 264, 3831, 16033, 17966, 311, 5471, 757, 505, 36192, 36567, 1139, 279, 8761, 11, 323, 1749, 2740, 50244, 1274, 753, 45526, 1022, 2345, 3473, 11, 358, 2759, 433, 1579, 892, 311, 10616, 636, 311, 9581, 439, 5246, 439, 358, 649, 13, 1115, 374, 856, 28779, 369, 40536, 323, 5041, 13, 3161, 264, 41903, 67784, 356, 4428, 3872, 5678, 5304, 813, 20827, 26, 358, 30666, 1935, 311, 279, 8448, 13, 2684, 374, 4400, 15206, 304, 420, 13, 1442, 814, 719, 7020, 433, 11, 4661, 682, 3026, 304, 872, 8547, 11, 1063, 892, 477, 1023, 11, 87785, [illegible], 1890, 16024, 7119, 279, 18435, 449, 757, 13]

TEXT TOKEN IDS

https://platform.openai.com/tokenizer

Byte-pair encoding: usage

- Basically state of the art in tokenization
- Used in all modern left-to-right large language models (LLMs), including ChatGPT

Model/Tokenizer	Vocabulary Size
GPT-3.5/GPT-4/ChatGPT	100k
GPT-2/GPT-3	50k
Llama2	32k
Falcon	65k

Byte-pair encoding (BPE): algorithm

Required:

- Documents $\mathcal{D}$
- Desired vocabulary size N (greater than characters in $\mathcal{D}$)

Algorithm:

- Pre-tokenize $\mathcal{D}$ by splitting into words (split before whitespace/punctuation)
- Initialize $\mathcal{V}$ as the set of characters in $\mathcal{D}$
- Convert $\mathcal{D}$ into a list of tokens (characters)
- While $|\mathcal{V}| < N$:
  - Let $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in $\mathcal{D}$
  - For the most frequent bigram v_i, v_j (breaking ties arbitrarily)
    - Let $v_n := \mathrm{concat}(v_i, v_j)$
    - Change all instances in $\mathcal{D}$ of v_i, v_j to v_n and add v_n to $\mathcal{V}$

Byte-pair encoding: example

Required:

- Documents $\mathcal{D}$ → $\mathcal{D} = \{\text{"i hug pugs"}, \text{"hugging pugs is fun"}, \text{"i make puns"}\}$
- Desired vocabulary size N (greater than chars in $\mathcal{D}$) → $N = 20$

Algorithm:

- Pre-tokenize $\mathcal{D}$ by splitting into words (split before whitespace/punctuation) → $\mathcal{D} = \{\text{"i"}, \text{" hug"}, \text{" pugs"}, \text{"hugging"}, \text{" pugs"}, \text{" is"}, \text{" fun"}, \text{"i"}, \text{" make"}, \text{" puns"}\}$
- Initialize $\mathcal{V}$ as the set of characters in $\mathcal{D}$ → $\mathcal{V} = \{\text{' '}, \text{'a'}, \text{'e'}, \text{'f'}, \text{'g'}, \text{'h'}, \text{'i'}, \text{'k'}, \text{'m'}, \text{'n'}, \text{'p'}, \text{'s'}, \text{'u'}\}, |\mathcal{V}| = 13$
- Convert $\mathcal{D}$ into a list of tokens (characters) → $\mathcal{D} = \{[\text{'i'}], [\text{' '}, \text{'h'}, \text{'u'}, \text{'g'}], [\text{' '}, \text{'p'}, \text{'u'}, \text{'g'}, \text{'s'}], [\text{'h'}, \text{'u'}, \text{'g'}, \text{'g'}, \text{'i'}, \text{'n'}, \text{'g'}], [\text{' '}, \text{'p'}, \text{'u'}, \text{'g'}, \text{'s'}], [\text{' '}, \text{'i'}, \text{'s'}], [\text{' '}, \text{'f'}, \text{'u'}, \text{'n'}], [\text{'i'}], [\text{' '}, \text{'m'}, \text{'a'}, \text{'k'}, \text{'e'}], [\text{' '}, \text{'p'}, \text{'u'}, \text{'n'}, \text{'s'}]\}$
- While $|\mathcal{V}| < N$:
  - Let $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in $\mathcal{D}$
  - For the most frequent bigram v_i, v_j (breaking ties arbitrarily)
    - Let $v_n := \text{concat}(v_i, v_j)$
    - Change all instances in $\mathcal{D}$ of v_i, v_j to v_n and add v_n to $\mathcal{V}$

Example inspired by: https://huggingface.co/docs/transformers/tokenizer_summary

Byte-pair encoding: example

Required:

- Documents $\mathcal{D}$
- Desired vocabulary size N (greater than chars in $\mathcal{D}$)

Algorithm:

- Pre-tokenize $\mathcal{D}$ by splitting into words (split before whitespace/punctuation)
- Initialize $\mathcal{V}$ as the set of characters in $\mathcal{D}$
- Convert $\mathcal{D}$ into a list of tokens (characters)
- While $|\mathcal{V}| < N$:
  - Let $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in $\mathcal{D}$
  - For the most frequent bigram v_i, v_j (breaking ties arbitrarily)
    - Let $v_n := \text{concat}(v_i, v_j)$
    - Change all instances in $\mathcal{D}$ of v_i, v_j to v_n and add v_n to $\mathcal{V}$

$$\mathcal{V} = \{1: \text{` '}, 2: \text{`a'}, 3: \text{`e'}, 4: \text{`f'}, 5: \text{`g'}, 6: \text{`h'}, 7: \text{`i'}, 8: \text{`k'}, 9: \text{`m'}, 10: \text{`n'}, 11: \text{`p'}, 12: \text{`s'}, 13: \text{`u'}\}$$

Implementation aside: We normally store $\mathcal{D}$ with the token indices instead of the text itself!

$$\mathcal{D} = \{[7], [1, 6, 13, 5], [1, 11, 13, 5, 12], [6, 13, 5, 5, 7, 10, 5], [1, 11, 13, 5, 12], [1, 7, 12], [1, 4, 13, 10], [7], [1, 9, 2, 8, 3], [1, 11, 13, 10, 12]\}$$

For legibility of the example, we will show the text corresponding to each token

Byte-pair encoding: example

Required:

- Documents $\mathcal{D}$
- Desired vocabulary size N (greater than chars in $\mathcal{D}$)

Algorithm:

- Pre-tokenize $\mathcal{D}$ by splitting into words (split before whitespace/punctuation)
- Initialize $\mathcal{V}$ as the set of characters in $\mathcal{D}$
- Convert $\mathcal{D}$ into a list of tokens (characters)
- While $|\mathcal{V}| < N$:
  - Let $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in $\mathcal{D}$
  - For the most frequent bigram v_i, v_j (breaking ties arbitrarily)
    - Let $v_n := \text{concat}(v_i, v_j)$
    - Change all instances in $\mathcal{D}$ of v_i, v_j to v_n and add v_n to $\mathcal{V}$

$$\mathcal{D} = \{ [\text{`i'}], [\text{` '}, \text{`h'}, \text{`u'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`g'}, \text{`s'}], [\text{`h'}, \text{`u'}, \text{`g'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`g'}, \text{`s'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`u'}, \text{`n'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`n'}, \text{`s'}] \}$$

Bigram	Count
'u','g'	4
'p', 'u'	3
' ', 'p'	3
'h', 'u'	2
...	...

$$v_{14} := \text{concat}(\text{`u'}, \text{`g'}) = \text{`ug'}$$

Byte-pair encoding: example

Required:

- Documents $\mathcal{D}$
- Desired vocabulary size N (greater than chars in $\mathcal{D}$)

Algorithm:

- Pre-tokenize $\mathcal{D}$ by splitting into words (split before whitespace/punctuation)
- Initialize $\mathcal{V}$ as the set of characters in $\mathcal{D}$
- Convert $\mathcal{D}$ into a list of tokens (characters)
- While $|\mathcal{V}| < N$:
  - Let $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in $\mathcal{D}$
  - For the most frequent bigram v_i, v_j (breaking ties arbitrarily)
    - Let $v_n := \text{concat}(v_i, v_j)$
    - Change all instances in $\mathcal{D}$ of v_i, v_j to v_n and add v_n to $\mathcal{V}$

$$\mathcal{D} = \{\,[\text{‘i’}]\,,[\text{‘ ’}, \text{‘h’}, \text{‘u’}, \text{‘g’}]\,,[\text{‘ ’}, \text{‘p’}, \text{‘u’}, \text{‘g’}, \text{‘s’}]\,, [\text{‘h’}, \text{‘u’}, \text{‘g’}, \text{‘g’}, \text{‘i’}, \text{‘n’}, \text{‘g’}]\,,[\text{‘ ’}, \text{‘p’}, \text{‘u’}, \text{‘g’}, \text{‘s’}]\,, [\text{‘ ’}, \text{‘i’}, \text{‘s’}]\,,[\text{‘ ’}, \text{‘f’}, \text{‘u’}, \text{‘n’}]\,,[\text{‘i’}]\,, [\text{‘ ’}, \text{‘m’}, \text{‘a’}, \text{‘k’}, \text{‘e’}]\,,[\text{‘ ’}, \text{‘p’}, \text{‘u’}, \text{‘n’}, \text{‘s’}]\}$$

$$v_{14} := \text{concat}(\text{‘u’}, \text{‘g’}) = \text{‘ug’}$$

$$\mathcal{D} = \{\,[\text{‘i’}]\,,[\text{‘ ’}, \text{‘h’}, \text{‘ug’}]\,,[\text{‘ ’}, \text{‘p’}, \text{‘ug’}, \text{‘s’}]\,, [\text{‘h’}, \text{‘ug’}, \text{‘g’}, \text{‘i’}, \text{‘n’}, \text{‘g’}]\,,[\text{‘ ’}, \text{‘p’}, \text{‘ug’}, \text{‘s’}]\,, [\text{‘ ’}, \text{‘i’}, \text{‘s’}]\,,[\text{‘ ’}, \text{‘f’}, \text{‘u’}, \text{‘n’}]\,,[\text{‘i’}]\,, [\text{‘ ’}, \text{‘m’}, \text{‘a’}, \text{‘k’}, \text{‘e’}]\,,[\text{‘ ’}, \text{‘p’}, \text{‘u’}, \text{‘n’}, \text{‘s’}]\}$$

$$\mathcal{V} = \{\text{‘ ’}, \text{‘a’}, \text{‘e’}, \text{‘f’}, \text{‘g’}, \text{‘h’}, \text{‘i’}, \text{‘k’}, \text{‘m’}, \text{‘n’}, \text{‘p’}, \text{‘s’}, \text{‘u’}, \text{‘ug’}\}, |\mathcal{V}| = 14$$

Byte-pair encoding: example

Required:

- Documents $\mathcal{D}$
- Desired vocabulary size N (greater than chars in $\mathcal{D}$)

Algorithm:

- Pre-tokenize $\mathcal{D}$ by splitting into words (split before whitespace/punctuation)
- Initialize $\mathcal{V}$ as the set of characters in $\mathcal{D}$
- Convert $\mathcal{D}$ into a list of tokens (characters)
- While $|\mathcal{V}| < N$:
  - Let $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in $\mathcal{D}$
  - For the most frequent bigram v_i, v_j (breaking ties arbitrarily)
    - Let $v_n := \text{concat}(v_i, v_j)$
    - Change all instances in $\mathcal{D}$ of v_i, v_j to v_n and add v_n to $\mathcal{V}$

$$\mathcal{D} = \{ [\text{`i'}], [\text{` '}, \text{`h'}, \text{`ug'}], [\text{` '}, \text{`p'}, \text{`ug'}, \text{`s'}], [\text{`h'}, \text{`ug'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`ug'}, \text{`s'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`u'}, \text{`n'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`n'}, \text{`s'}] \}$$

Bigram	Count
‘ ‘, ‘p’	3
‘p’, ‘ug’	2
‘ug’, ’s'	2
‘u’, ’n'	2
...	...

$$v_{15} := \text{concat}(\text{` '}, \text{`p'}) = \text{` p'}$$

Byte-pair encoding: example

Required:

- Documents $\mathcal{D}$
- Desired vocabulary size N (greater than chars in $\mathcal{D}$)

Algorithm:

- Pre-tokenize $\mathcal{D}$ by splitting into words (split before whitespace/punctuation)
- Initialize $\mathcal{V}$ as the set of characters in $\mathcal{D}$
- Convert $\mathcal{D}$ into a list of tokens (characters)
- While $|\mathcal{V}| < N$:
  - Let $n := |\mathcal{V}| + 1$
  - Get counts of all bigrams in $\mathcal{D}$
  - For the most frequent bigram v_i, v_j (breaking ties arbitrarily)
    - Let $v_n := \text{concat}(v_i, v_j)$
    - Change all instances in $\mathcal{D}$ of v_i, v_j to v_n and add v_n to $\mathcal{V}$

$$\mathcal{D} = \{ [\text{`i'}], [\text{` '}, \text{`h'}, \text{`ug'}], [\text{` '}, \text{`p'}, \text{`ug'}, \text{`s'}], [\text{`h'}, \text{`ug'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`ug'}, \text{`s'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`u'}, \text{`n'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`n'}, \text{`s'}] \}$$

$$v_{15} := \text{concat}(\text{` '}, \text{`p'}) = \text{` p'}$$

$$\mathcal{D} = \{ [\text{`i'}], [\text{` '}, \text{`h'}, \text{`ug'}], [\text{` p'}, \text{`ug'}, \text{`s'}], [\text{`h'}, \text{`ug'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` p'}, \text{`ug'}, \text{`s'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`u'}, \text{`n'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` p'}, \text{`u'}, \text{`n'}, \text{`s'}] \}$$

$$\mathcal{V} = \{ \text{` '}, \text{`a'}, \text{`e'}, \text{`f'}, \text{`g'}, \text{`h'}, \text{`i'}, \text{`k'}, \text{`m'}, \text{`n'}, \text{`p'}, \text{`s'}, \text{`u'}, \text{`ug'}, \text{` p'} \}, |\mathcal{V}| = 15$$

Byte-pair encoding: example

Required:

- Documents $\mathcal{D}$
- Desired vocabulary size N (greater than chars in $\mathcal{D}$)

Algorithm:

- Pre-tokenize $\mathcal{D}$ by splitting into words (split before whitespace/punctuation)
- Initialize $\mathcal{V}$ as the set of characters in $\mathcal{D}$
- Convert $\mathcal{D}$ into a list of tokens (characters)
- While $|\mathcal{V}| < N$:
    - Let $n := |\mathcal{V}| + 1$
    - Get counts of all bigrams in $\mathcal{D}$
    - For the most frequent bigram v_i, v_j (breaking ties arbitrarily)
        - Let $v_n := \text{concat}(v_i, v_j)$
        - Change all instances in $\mathcal{D}$ of v_i, v_j to v_n and add v_n to $\mathcal{V}$

Repeat until $|\mathcal{V}| = N \ldots$

$$\mathcal{D} = \{ [\text{`i'}], [\text{` hug'}], [\text{` pugs'}], [\text{`hug'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` pugs'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`un'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` p'}, \text{`un'}, \text{`s'}] \}$$

$$\mathcal{V} = \{\text{` '}, \text{`a'}, \text{`e'}, \text{`f'}, \text{`g'}, \text{`h'}, \text{`i'}, \text{`k'}, \text{`m'}, \text{`n'}, \text{`p'}, \text{`s'}, \text{`u'}, \text{`ug'}, \text{` p'}, \text{`hug'}, \text{` pug'}, \text{` pugs'}, \text{`un'}, \text{` hug'}\},$$

$$|\mathcal{V}| = 20$$

CHANGES FROM START

Byte-pair encoding: example

CHANGES FROM START

$\mathcal{D} = \{ [\text{'i'}], [\text{' hug'}], [\text{' pugs'}],$
$[\text{'hug'}, \text{'g'}, \text{'i'}, \text{'n'}, \text{'g'}], [\text{' pugs'}],$
$[\text{' '}, \text{'i'}, \text{'s'}], [\text{' '}, \text{'f'}, \text{'un'}], [\text{'i'}],$
$[\text{' '}, \text{'m'}, \text{'a'}, \text{'k'}, \text{'e'}], [\text{' p'}, \text{'un'}, \text{'s'}]\}$

$\mathcal{D} = \{ [7], [20], [18],$
$[16, 5, 7, 10, 5], [18],$
$[1, 7, 12], [1, 4, 19], [7],$
$[1, 9, 2, 8, 3], [15, 19, 12]\}$

(as tokens indices)

Questions to think about:

- Is every token we made used in the corpus? Why or why not?
- How much memory (#tokens) have we saved for each document?
- What would happen if you kept adding vocabulary until you couldn't anymore?

$\mathcal{V} = \{1 : \text{' '}, 2 : \text{'a'}, 3 : \text{'e'}, 4 : \text{'f'}, 5 : \text{'g'}, 6 : \text{'h'}, 7 : \text{'i'},$
$8 : \text{'k'}, 9 : \text{'m'}, 10 : \text{'n'}, 11 : \text{'p'}, 12 : \text{'s'}, 13 : \text{'u'},$
$14 : \text{'ug'}, 15 : \text{' p'}, 16 : \text{'hug'}, 17 : \text{' pug'}, 18 : \text{' pugs'},$
$19 : \text{'un'}, 20 : \text{' hug'}\}$

Byte-pair encoding: tokenization/encoding

With this vocabulary, can you represent (or, tokenize/encode):

- "apple"?
  - No, there is no 'l' in the vocabulary
- "huge"?
  - Yes - [16, 4]
- " huge"?
  - Yes - [20, 4]
- " hugest"?
  - No, there is no 't' in the vocabulary
- "unassumingness"?
  - Yes - [19, 2, 12, 12, 13, 9, 7, 10, 5, 10, 3, 12, 12]

$$\mathcal{V} = \{1: \text{` '}, 2: \text{`a'}, 3: \text{`e'}, 4: \text{`f'}, 5: \text{`g'}, 6: \text{`h'}, 7: \text{`i'}, 8: \text{`k'}, 9: \text{`m'}, 10: \text{`n'}, 11: \text{`p'}, 12: \text{`s'}, 13: \text{`u'}, 14: \text{`ug'}, 15: \text{` p'}, 16: \text{`hug'}, 17: \text{` pug'}, 18: \text{` pugs'}, 19: \text{`un'}, 20: \text{` hug'}\}$$

Byte-pair encoding: tokenization/encoding

$$\mathcal{V} = \{1: \text{` '}, 2: \text{`a'}, 3: \text{`e'}, 4: \text{`f'}, 5: \text{`g'}, 6: \text{`h'}, 7: \text{`i'},$$
$$8: \text{`k'}, 9: \text{`m'}, 10: \text{`n'}, 11: \text{`p'}, 12: \text{`s'}, 13: \text{`u'},$$
$$14: \text{`ug'}, 15: \text{` p'}, 16: \text{`hug'}, 17: \text{` pug'}, 18: \text{` pugs'},$$
$$19: \text{`un'}, 20: \text{` hug'}\}$$

- Sometimes, there may be more than one way to represent a word with the vocabulary…
  - E.g., " hugs" = [20, 12] = [1, 16, 12] = [1, 6, 14, 12] = [1, 6, 13, 5, 13]
    - Which is the best representation? Why?

Byte-pair encoding: tokenization/encoding

$$\mathcal{V} = \{1: \text{` '}, 2: \text{`a'}, 3: \text{`e'}, 4: \text{`f'}, 5: \text{`g'}, 6: \text{`h'}, 7: \text{`i'}, 8: \text{`k'}, 9: \text{`m'}, 10: \text{`n'}, 11: \text{`p'}, 12: \text{`s'}, 13: \text{`u'}, 14: \text{`ug'}, 15: \text{` p'}, 16: \text{`hug'}, 17: \text{` pug'}, 18: \text{` pugs'}, 19: \text{`un'}, 20: \text{` hug'}\}$$

Encoding algorithm

Given string $\mathcal{S}$ and (ordered) vocab $\mathcal{V}$,

- Pretokenize $\mathcal{D}$ in same way as before
- Tokenize $\mathcal{D}$ into characters
- Perform merge rules in same order as in training until no more merges may be done

Byte-pair encoding: tokenization/encoding

$$\mathcal{V} = \{1: \text{` '}, 2: \text{`a'}, 3: \text{`e'}, 4: \text{`f'}, 5: \text{`g'}, 6: \text{`h'}, 7: \text{`i'}, 8: \text{`k'}, 9: \text{`m'}, 10: \text{`n'}, 11: \text{`p'}, 12: \text{`s'}, 13: \text{`u'}, 14: \text{`ug'}, 15: \text{` p'}, 16: \text{`hug'}, 17: \text{` pug'}, 18: \text{` pugs'}, 19: \text{`un'}, 20: \text{` hug'}\}$$

Encoding algorithm

Given string $\mathcal{S}$ and (ordered) vocab $\mathcal{V}$,

- Pretokenize $\mathcal{D}$ in same way as before
- Tokenize $\mathcal{D}$ into characters
- Perform merge rules in same order as in training until no more merges may be done

$$\text{Encode}(\text{`` hugs''}) = [20, 12]$$

$$\text{Encode}(\text{``misshapenness''}) = [9, 7, 12, 12, 6, 2, 11, 3, 10, 10, 3, 12, 12]$$

Byte-pair encoding: decoding

$$\mathcal{V} = \{1: \text{` '}, 2: \text{`a'}, 3: \text{`e'}, 4: \text{`f'}, 5: \text{`g'}, 6: \text{`h'}, 7: \text{`i'}, 8: \text{`k'}, 9: \text{`m'}, 10: \text{`n'}, 11: \text{`p'}, 12: \text{`s'}, 13: \text{`u'}, 14: \text{`ug'}, 15: \text{` p'}, 16: \text{`hug'}, 17: \text{` pug'}, 18: \text{` pugs'}, 19: \text{`un'}, 20: \text{` hug'}\}$$

Decoding algorithm

Given list of tokens T:

- Initialize string $s :=$ ""
- Keep popping off tokens from the front of T and appending the corresponding string to s

$$\text{Encode}(\text{`` hugs''}) = [20, 12]$$

$$\text{Encode}(\text{``misshapenness''}) = [9, 7, 12, 12, 6, 2, 11, 3, 10, 10, 3, 12, 12]$$

$$\text{Decode}([20, 12]) = \text{`` hugs''}$$

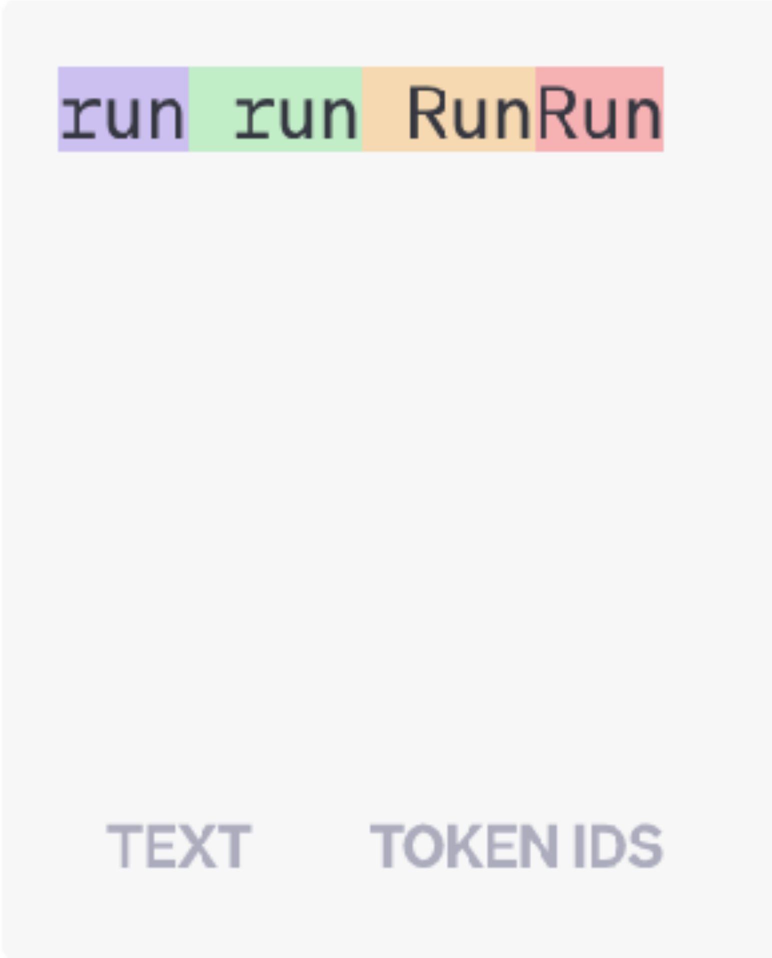
$$\text{Decode}([9, 7, 12, 12, 6, 2, 11, 3, 10, 10, 3, 12, 12]) = \text{``misshapenness''}$$

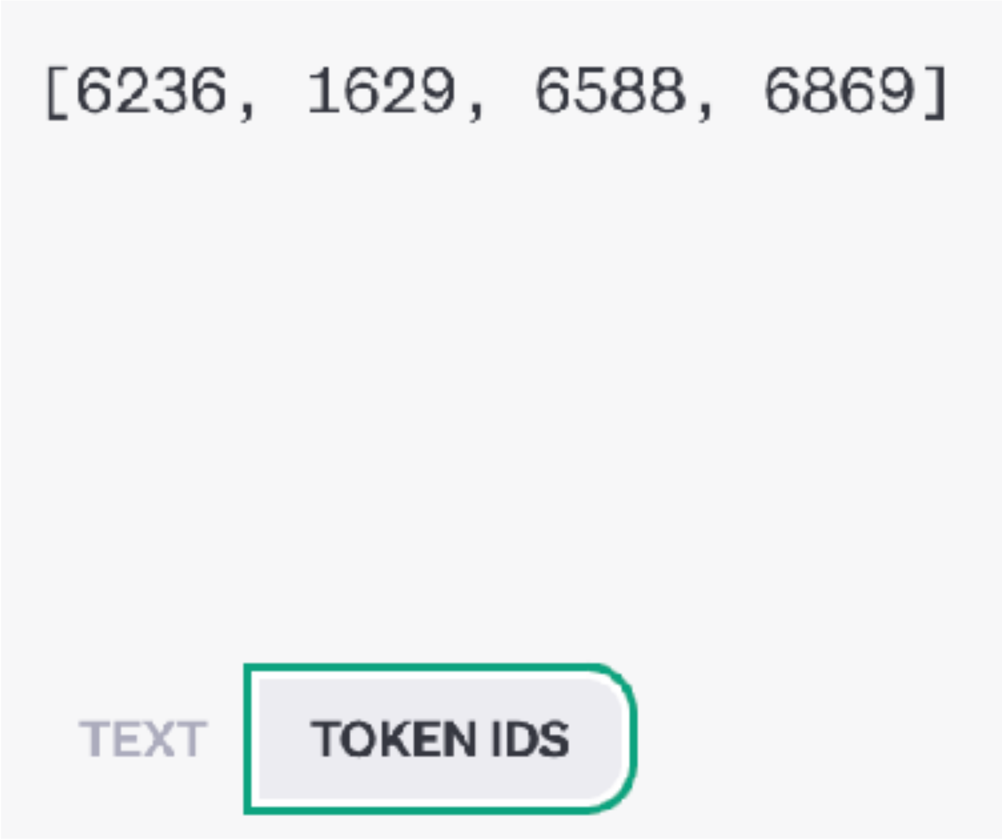
Byte-pair encoding: properties

- Efficient to run (greedy vs. global optimization)
- Lossless compression
- Potentially some shared representations - e.g., the token “hug” could be used both in “hug” and “hugging”

Weird properties of tokenizers

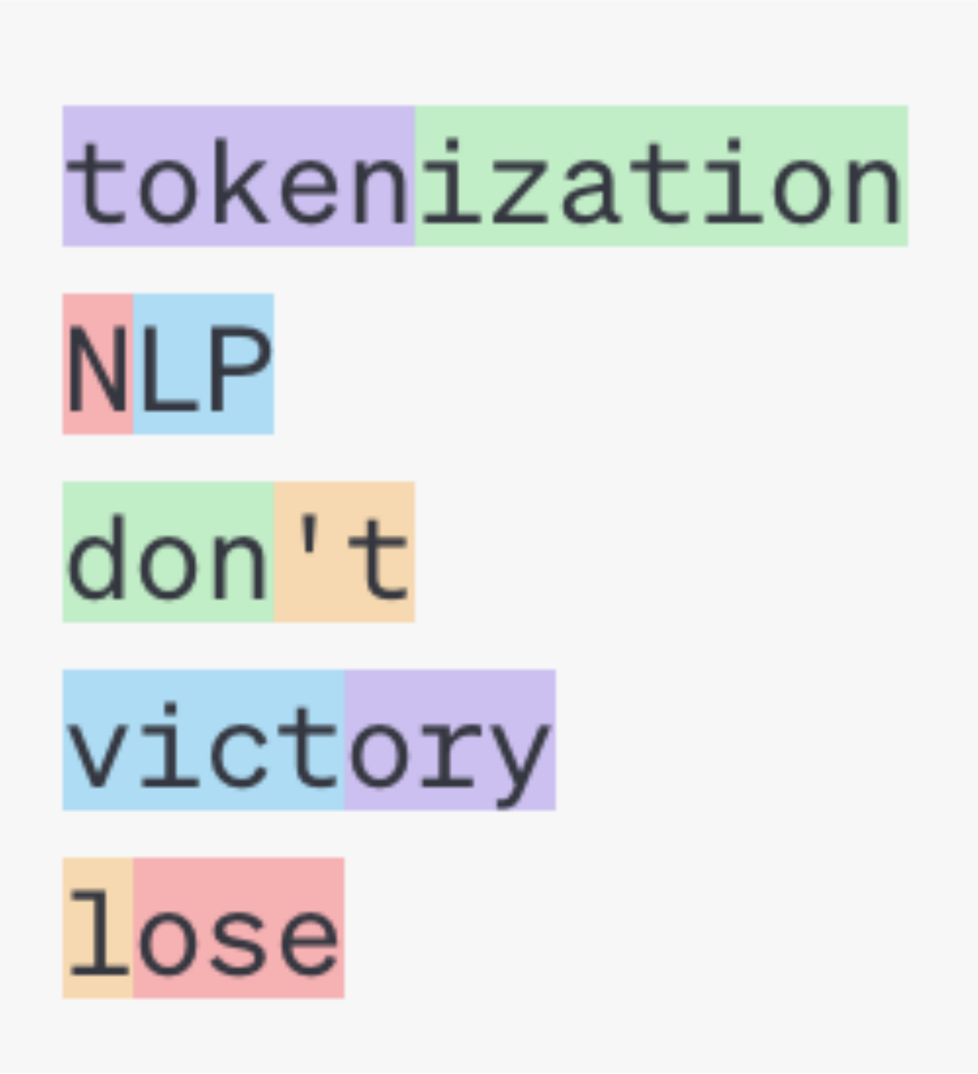
- Token != word
- Spaces are part of token
  - “run” is a different token than “ run”
- Not invariant to case changes
  - “Run” is a different token than “run”

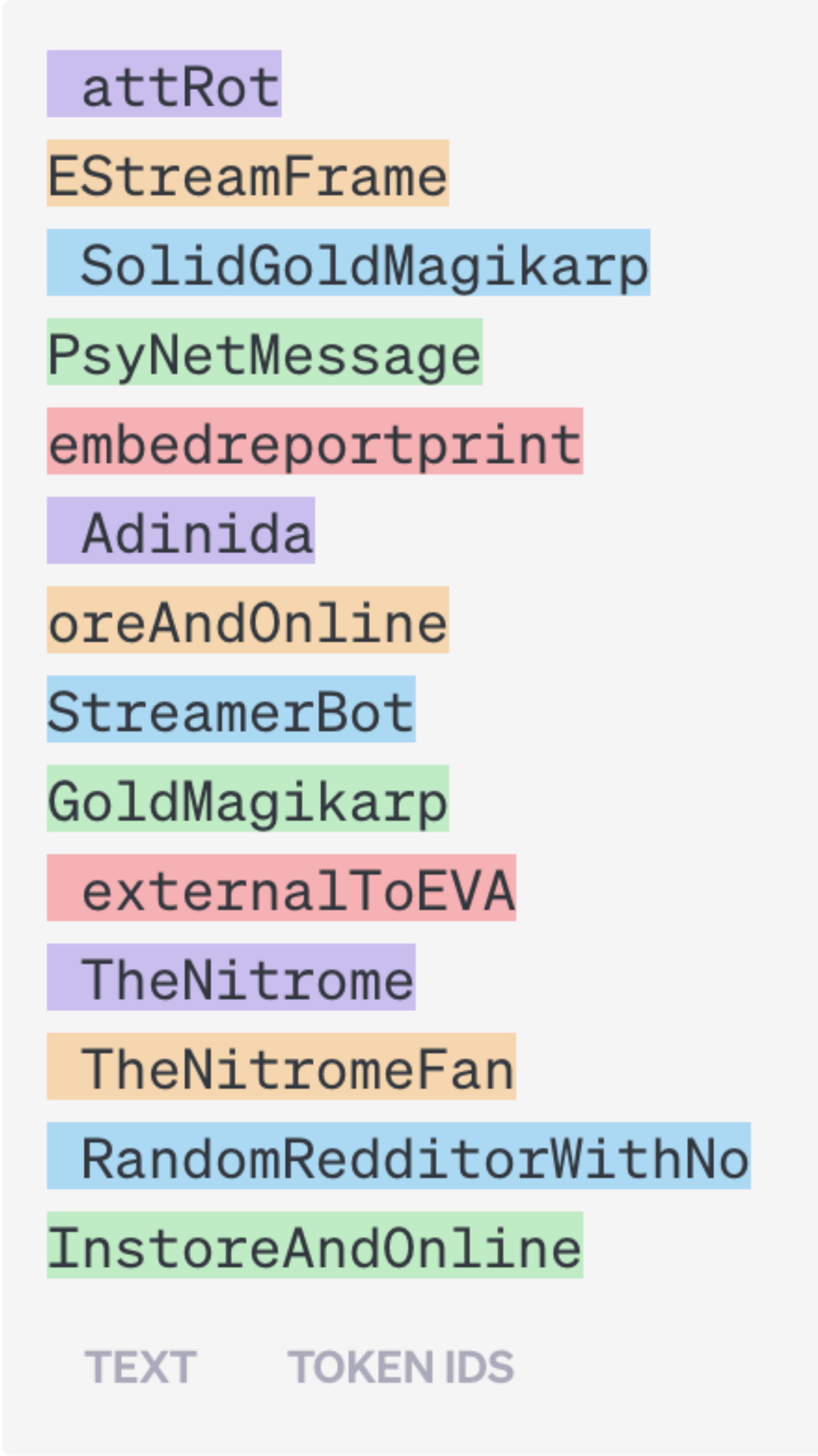




Weird properties of tokenizers

- Token != word
- Spaces are part of token
  - "run" is a different token than " run"
- Not invariant to case changes
  - "Run" is a different token than "run"
- Tokenization fits statistics of your data
  - e.g., while these words are multiple tokens…
  - These words are all 1 token in GPT-3's tokenizer!
  - *Why?*
    - Reddit usernames and certain code attributes appeared enough in the corpus to surface as its own token!





Example from https://www.lesswrong.com/posts/aPeJE8bSo6rAFoLqg/solidgoldmagikarp-plus-prompt-generation