Text Tokenization in Language Models

Ling 282/482: Deep Learning for Computational Linguistics

C.M. Downey

Fall 2025

Traditional Tokenization

Traditional Tokenization

- “Hello world. I can’t wait.” → [Hello, world, ., I, can, n’t, wait, .]

Traditional Tokenization

- “Hello world. I can’t wait.” → [Hello, world, ., I, can, n’t, wait, .]
- **Word level**: the goal is to split text into “words”. This is mostly determined by **whitespace**
  - Punctuation is separated out

Traditional Tokenization

- "Hello world. I can't wait." → [Hello, world, ., I, can, n't, wait, .]
- **Word level**: the goal is to split text into "words". This is mostly determined by **whitespace**
  - Punctuation is separated out
- **Language dependent**: relies on arbitrary rules for **specific languages**.
  - e.g. tokenize **English contractions** as separate words "can" + "n't"
  - Each language has own rules: こんにちは世界。→ [こんにちは, 世界, 。]

Traditional Tokenization

- “Hello world. I can’t wait.” → [Hello, world, ., I, can, n’t, wait, .]
- **Word level**: the goal is to split text into “words”. This is mostly determined by **whitespace**
  - Punctuation is separated out
- **Language dependent**: relies on arbitrary rules for **specific languages**.
  - e.g. tokenize **English contractions** as separate words “can” + “n’t”
  - Each language has own rules: こんにちは世界。→ [こんにちは, 世界, 。]
- **Not reversible**: some **information is lost** during tokenization
  - “Hello world.” vs. “Hello   world.” vs. “Hello world .”

Traditional Tokenization

Traditional Tokenization

- **Out-of-Vocabulary (OOV)**: model is trained to cover a **word-level vocabulary**, and **can't handle new words** after training
  - e.g. model is set up to handle 50k words. Any new words are converted to a symbol for **unknown tokens <unk>**
  - Model **can't adapt** to novel expressions (e.g. "skibidi")

Traditional Tokenization

- **Out-of-Vocabulary (OOV)**: model is trained to cover a **word-level vocabulary**, and **can't handle new words** after training
  - e.g. model is set up to handle 50k words. Any new words are converted to a symbol for **unknown tokens <unk>**
  - Model **can't adapt** to novel expressions (e.g. "skibidi")
- **Large vocabulary size**: a single language might have **hundreds of thousands** of words
  - In a neural model, this is costly at the **embedding** and **softmax layers**

Alternatives

Alternatives

- **Character tokenization**: split text into **individual characters**
  - **Advantages**: small vocab size, low chance of OOV, somewhat language-general
  - **Drawbacks**: much harder modeling, longer sequences, not efficient at handling repeated n-grams, can still have OOV characters

Alternatives

- **Character tokenization**: split text into **individual characters**
  - **Advantages**: small vocab size, low chance of OOV, somewhat language-general
  - **Drawbacks**: much harder modeling, longer sequences, not efficient at handling repeated n-grams, can still have OOV characters
- **“Sub-word” tokenization**: split text into **variable-sized units**
  - Optimizes sequence length while **avoiding OOV**
  - Almost **universally used** in LMs today

Subword Tokenization

Subword Tokenization

- Key idea: **frequent** sequences grouped into a **single token**, while **rare** sequences are tokenized as **characters** or **smaller chunks**
  - “I can’t wait.” → [I, ca, #n’t, wait, #.]

Subword Tokenization

- Key idea: **frequent** sequences grouped into a **single token**, while **rare** sequences are tokenized as **characters** or **smaller chunks**
  - "I can't wait." → [I, ca, #n't, wait, #.]
- Vocabulary minimally contains **all* characters in training set**
  - This **avoids OOV**: an unseen word can always be tokenized into characters
  - "skibidi" → [s, k, i, b, i, d, i]
  - *Can exclude very rare characters if desired

Subword Tokenization

- Key idea: **frequent** sequences grouped into a **single token**, while **rare** sequences are tokenized as **characters** or **smaller chunks**
  - “I can’t wait.” → [I, ca, #n’t, wait, #.]
- Vocabulary minimally contains **all* characters in training set**
  - This **avoids OOV**: an unseen word can always be tokenized into characters
  - “skibidi” → [s, k, i, b, i, d, i]
  - *Can exclude very rare characters if desired
- **Vocabulary size** is treated as a **hyper-parameter**
  - i.e. practitioner chooses size, and an algorithm devises the tokenization rules

Varieties of Subword Tokenization

Varieties of Subword Tokenization

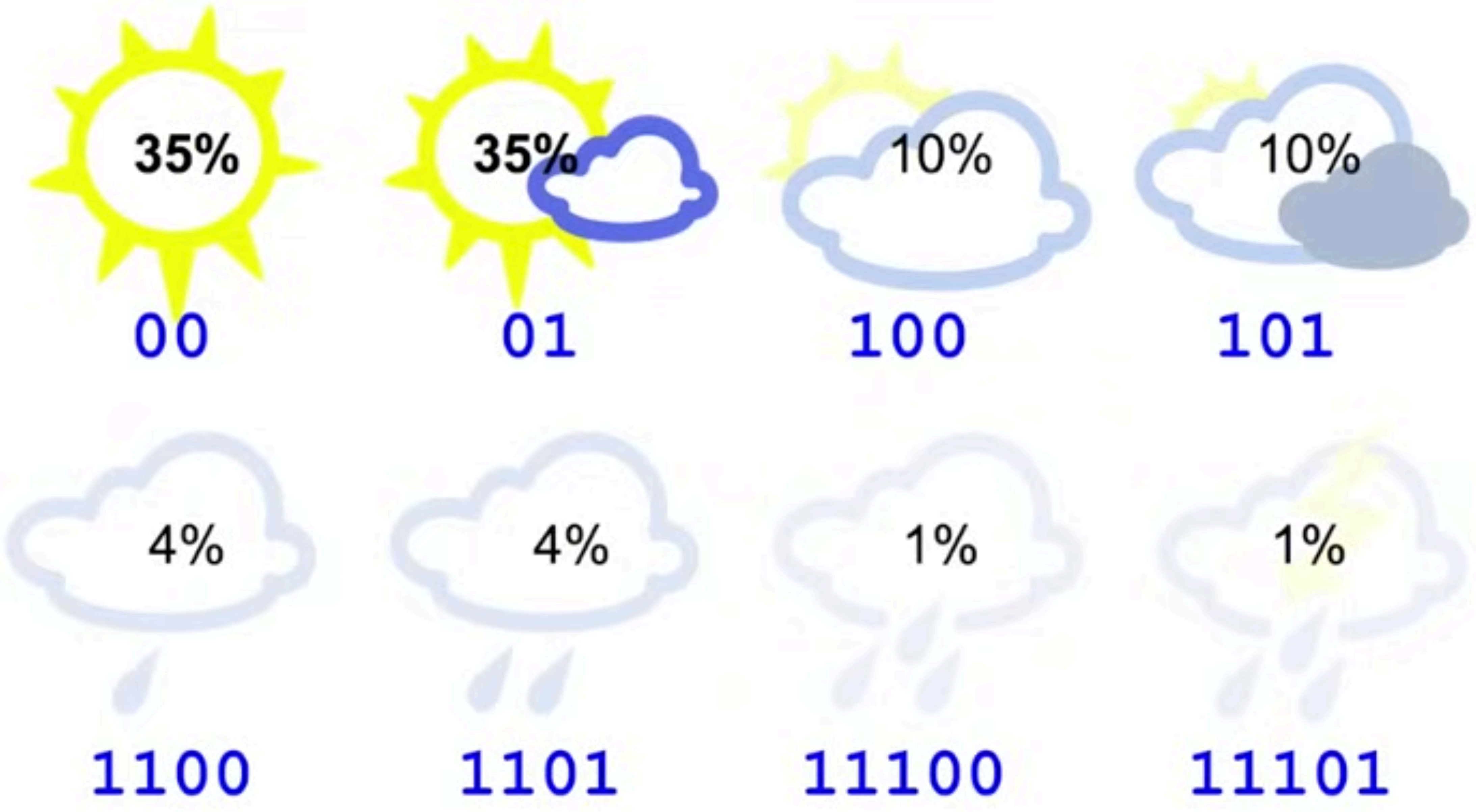
- The tokenization used in modern LMs is variably referred to as **Byte Pair Encoding (BPE)**, **SentencePiece**, **WordPiece**, or sometimes just **"subword tokenization"**

Varieties of Subword Tokenization

- The tokenization used in modern LMs is variably referred to as **Byte Pair Encoding (BPE)**, **SentencePiece**, **WordPiece**, or sometimes just **“subword tokenization”**
- These terms are **not interchangeable**, though they are sometimes casually used that way
  - There is significant overlap between these terms, but also nuanced differences
  - We’ll cover what each of these specifically refers to

Byte Pair Encoding (BPE)

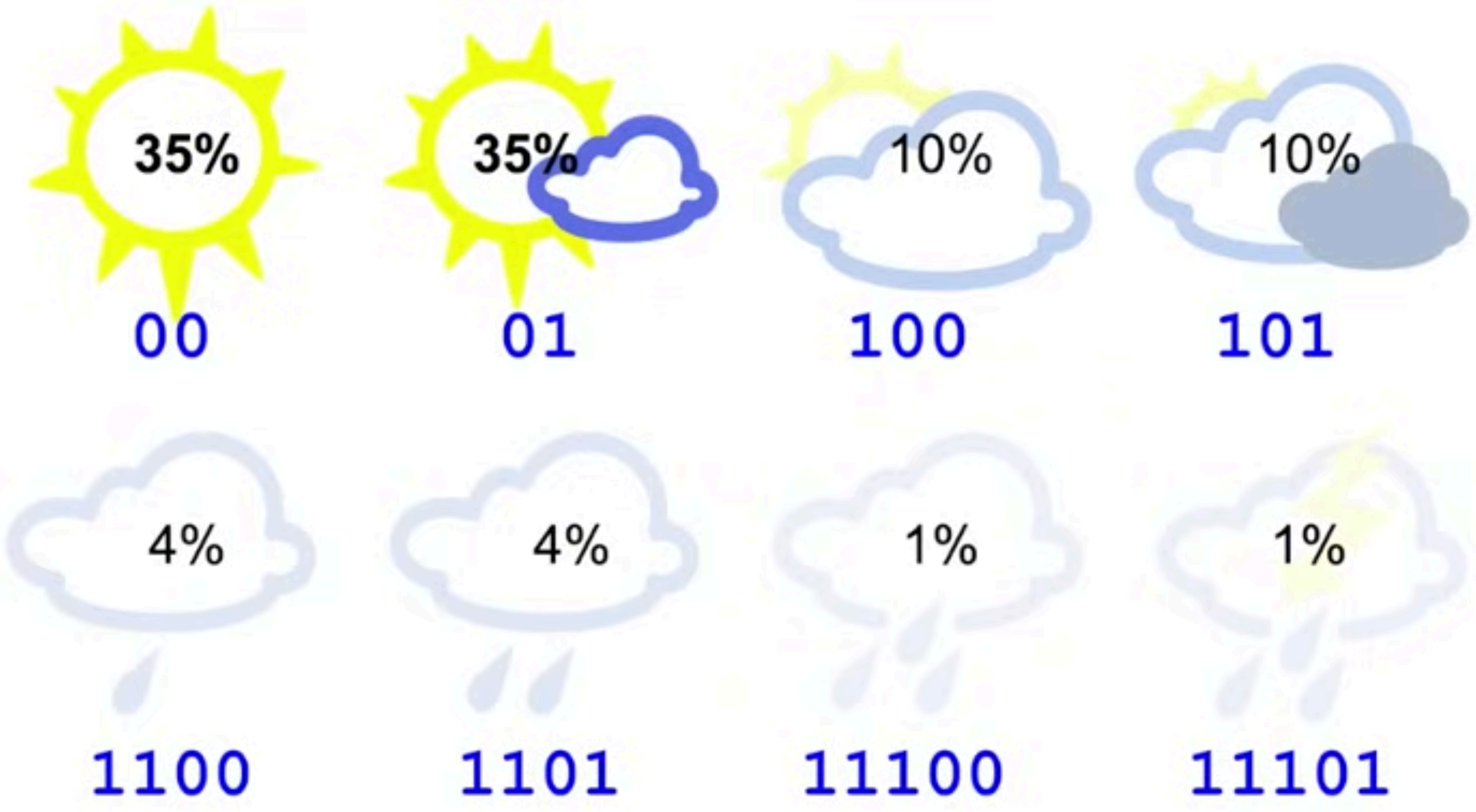
Byte Pair Encoding (BPE)



source

Byte Pair Encoding (BPE)

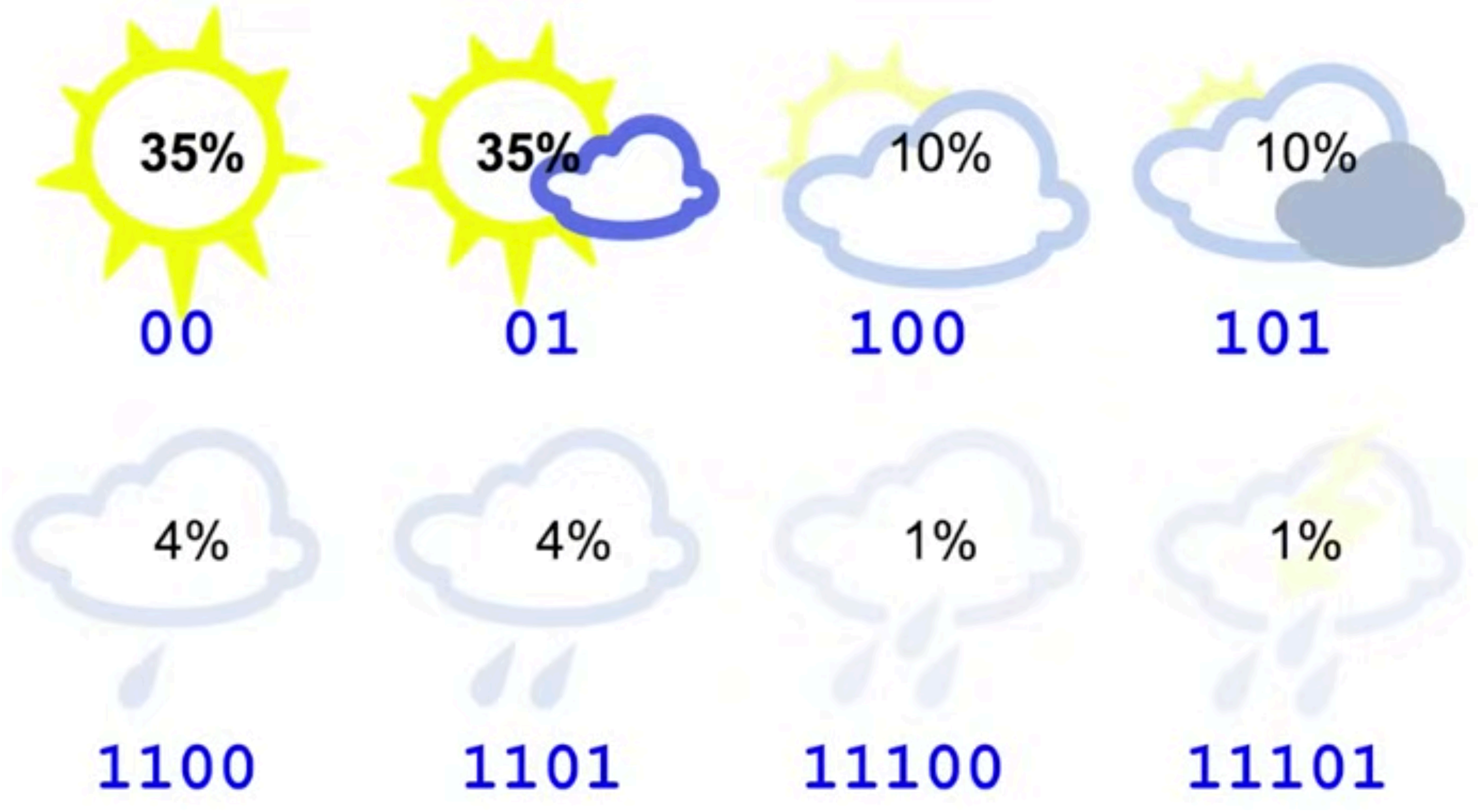
- Formalized for NLP in Sennrich, Haddow, and Birch (2016)
  - Implements an algorithm first proposed in 1994
  - Developed for **Neural Machine Translation** (NMT)



source

Byte Pair Encoding (BPE)

- Formalized for NLP in Sennrich, Haddow, and Birch (2016)
  - Implements an algorithm first proposed in 1994
  - Developed for **Neural Machine Translation** (NMT)
- Based on optimal codes from **Information Theory**
  - Frequent sequences are encoded with **fewer symbols** (tokens)
  - Rare seqeunces are encoded with **more symbols** (tokens)
  - This **optimizes overall code length** (number of tokens per sentence)



source

Byte Pair Encoding (BPE)

Byte Pair Encoding (BPE)

- Example data: “Stronger, faster, better.”

Byte Pair Encoding (BPE)

- Example data: “Stronger, faster, better.”
- Vocabulary is initialized at the **character level**
  - Initial tokenization: [S t r o n g e r , f a s t e r , b e t t e r .]

Byte Pair Encoding (BPE)

- Example data: “Stronger, faster, better.”
- Vocabulary is initialized at the **character level**
  - Initial tokenization: [S t r o n g e r , f a s t e r , b e t t e r .]
- At each step:
  - Count all **character pairs** in the data to find the **most frequent pair**
  - Combine the most frequent pair into a **single symbol** (here: (“e”, “r”) → “er”). **Add** the new symbol to the vocabulary.
  - New tokenization: [S t r o n g er , f a s t er , b e t t er .]

Byte Pair Encoding (BPE)

- Example data: “Stronger, faster, better.”
- Vocabulary is initialized at the **character level**
  - Initial tokenization: [S t r o n g e r , f a s t e r , b e t t e r .]
- At each step:
  - Count all **character pairs** in the data to find the **most frequent pair**
  - Combine the most frequent pair into a **single symbol** (here: (“e”, “r”) → “er”). **Add** the new symbol to the vocabulary.
  - New tokenization: [S t r o n g er , f a s t er , b e t t er .]
- **Repeat** this step until the **chosen vocabulary size is reached**

BPE Nuances

BPE Nuances

- Symbol pairs that **cross word boundaries** are **NOT counted**
  - If we have the words “cat tail”, (t, t) is **not** a valid symbol pair
  - The original implementation assumes the input has **already been segmented into “words”**
    - This is mainly for **efficiency reasons** (algorithm is still sound without pre-segmentation)

BPE Nuances

- Symbol pairs that **cross word boundaries** are **NOT counted**
  - If we have the words “cat tail”, (t, t) is **not** a valid symbol pair
  - The original implementation assumes the input has **already been segmented into “words”**
    - This is mainly for **efficiency reasons** (algorithm is still sound without pre-segmentation)
- Most **punctuation** marks are **tokenized separately** (as if they were separate words)

BPE Nuances

- Symbol pairs that **cross word boundaries** are **NOT counted**
  - If we have the words “cat tail”, (t, t) is **not** a valid symbol pair
  - The original implementation assumes the input has **already been segmented into “words”**
    - This is mainly for **efficiency reasons** (algorithm is still sound without pre-segmentation)
- Most **punctuation** marks are **tokenized separately** (as if they were separate words)
- @ used to indicate **no space** between one token and the next
  - [s@ t@ r@ o@ n@ g@ er@ ,]
  - We will see that subsequent implementations use **different conventions**

Note on WordPiece

Feature	SentencePiece	subword-nmt	WordPiece
Supported algorithm	BPE, unigram, char, word	BPE	BPE*
OSS?	Yes	Yes	Google internal
Subword regularization	Yes	No	No
Python Library (pip)	Yes	No	N/A
C++ Library	Yes	No	N/A
Pre-segmentation required?	No	Yes	Yes
Customizable normalization (e.g., NFKC)	Yes	No	N/A
Direct id generation	Yes	No	N/A

Note that BPE algorithm used in WordPiece is slightly different from the original BPE.

source

Note on WordPiece

- **WordPiece** is (essentially) **Google's implementation of BPE**

Feature	SentencePiece	subword-nmt	WordPiece
Supported algorithm	BPE, unigram, char, word	BPE	BPE*
OSS?	Yes	Yes	Google internal
Subword regularization	Yes	No	No
Python Library (pip)	Yes	No	N/A
C++ Library	Yes	No	N/A
Pre-segmentation required?	No	Yes	Yes
Customizable normalization (e.g., NFKC)	Yes	No	N/A
Direct id generation	Yes	No	N/A

Note that BPE algorithm used in WordPiece is slightly different from the original BPE.

source

Note on WordPiece

- **WordPiece** is (essentially) **Google's implementation of BPE**
- There are some **subtle differences** algorithmically

Feature	SentencePiece	subword-nmt	WordPiece
Supported algorithm	BPE, unigram, char, word	BPE	BPE*
OSS?	Yes	Yes	Google internal
Subword regularization	Yes	No	No
Python Library (pip)	Yes	No	N/A
C++ Library	Yes	No	N/A
Pre-segmentation required?	No	Yes	Yes
Customizable normalization (e.g., NFKC)	Yes	No	N/A
Direct id generation	Yes	No	N/A

Note that BPE algorithm used in WordPiece is slightly different from the original BPE.

source

Note on WordPiece

- **WordPiece** is (essentially) **Google's implementation of BPE**
- There are some **subtle differences** algorithmically
- Both the original implementation (Sennrich et al.) and the later SentencePiece are **open source**, whereas WordPiece is **Google internal**

Feature	SentencePiece	subword-nmt	WordPiece
Supported algorithm	BPE, unigram, char, word	BPE	BPE*
OSS?	Yes	Yes	Google internal
Subword regularization	Yes	No	No
Python Library (pip)	Yes	No	N/A
C++ Library	Yes	No	N/A
Pre-segmentation required?	No	Yes	Yes
Customizable normalization (e.g., NFKC)	Yes	No	N/A
Direct id generation	Yes	No	N/A

Note that BPE algorithm used in WordPiece is slightly different from the original BPE.

source

Note on WordPiece

- **WordPiece** is (essentially) **Google's implementation of BPE**
- There are some **subtle differences** algorithmically
- Both the original implementation (Sennrich et al.) and the later SentencePiece are **open source**, whereas WordPiece is **Google internal**
- WordPiece won't be important to discuss further, but the term still gets thrown arround

Feature	SentencePiece	subword-nmt	WordPiece
Supported algorithm	BPE, unigram, char, word	BPE	BPE*
OSS?	Yes	Yes	Google internal
Subword regularization	Yes	No	No
Python Library (pip)	Yes	No	N/A
C++ Library	Yes	No	N/A
Pre-segmentation required?	No	Yes	Yes
Customizable normalization (e.g., NFKC)	Yes	No	N/A
Direct id generation	Yes	No	N/A

Note that BPE algorithm used in WordPiece is slightly different from the original BPE.

source

SentencePiece

SentencePiece

SentencePiece

- **SentencePiece**: a **software library** that implements subword tokenization

SentencePiece

- **SentencePiece**: a **software library** that implements subword tokenization
- Includes **BPE** as an algorithm option (and improves **efficiency**)

SentencePiece

- **SentencePiece**: a **software library** that implements subword tokenization
- Includes **BPE** as an algorithm option (and improves **efficiency**)
- Integrates several **other subword algorithms** (notably **“Unigram”**)

SentencePiece

- **SentencePiece**: a **software library** that implements subword tokenization
- Includes **BPE** as an algorithm option (and improves **efficiency**)
- Integrates several **other subword algorithms** (notably **“Unigram”**)
- Incorporates **regularization “tricks”** such as Subword Regularization and BPE Dropout

SentencePiece

- **SentencePiece**: a **software library** that implements subword tokenization
- Includes **BPE** as an algorithm option (and improves **efficiency**)
- Integrates several **other subword algorithms** (notably “**Unigram**”)
- Incorporates **regularization “tricks”** such as Subword Regularization and BPE Dropout
- More or less the **standard tokenization library** for LMs today

SentencePiece API

SentencePiece API

- Focuses on the **sentence** as the primary unit, rather than the word
  - **Whitespace** is treated as a **normal character**
  - “Hello world.” → [Hello _wor ld .]
  - (Underscore indicates **leading whitespace**)

SentencePiece API

- Focuses on the **sentence** as the primary unit, rather than the word
  - **Whitespace** is treated as a **normal character**
  - “Hello world.” → [Hello _wor ld .]
  - (Underscore indicates **leading whitespace**)
- Emphasizes “**lossless**” **tokenization**
  - `raw_text = detokenize(tokenize(raw_text))`

SentencePiece API

- Focuses on the **sentence** as the primary unit, rather than the word
  - **Whitespace** is treated as a **normal character**
  - “Hello world.” → [Hello _wor ld .]
  - (Underscore indicates **leading whitespace**)
- Emphasizes **“lossless” tokenization**
  - `raw_text = detokenize(tokenize(raw_text))`
- **Does not assume** the text is **pre-segmented** into “words”
  - Languages with spaces between words (English, Hindi, Russian, etc.) are treated the same as those without (Chinese, Japanese, Thai)

“Unigram Language Model” segmentation

“Unigram Language Model” segmentation

- SentencePiece incorporates Unigram as an **alternative to BPE**

“Unigram Language Model” segmentation

- SentencePiece incorporates Unigram as an **alternative to BPE**
- Vocabulary size still a hyperparameter, but the algorithm is **quite different**

“Unigram Language Model” segmentation

- SentencePiece incorporates Unigram as an **alternative to BPE**
- Vocabulary size still a hyperparameter, but the algorithm is **quite different**
  - Instead of growing the vocab by merging symbols, **shrink vocab by “pruning”**

“Unigram Language Model” segmentation

- SentencePiece incorporates Unigram as an **alternative to BPE**
- Vocabulary size still a hyperparameter, but the algorithm is **quite different**
  - Instead of growing the vocab by merging symbols, **shrink vocab by “pruning”**
  - Large **“seed vocabulary”** is formed from common character-sequences

“Unigram Language Model” segmentation

- SentencePiece incorporates Unigram as an **alternative to BPE**
- Vocabulary size still a hyperparameter, but the algorithm is **quite different**
  - Instead of growing the vocab by merging symbols, **shrink vocab by “pruning”**
  - Large **“seed vocabulary”** is formed from common character-sequences
  - The **overall frequency** of each item is calculated (**unigram probability**)

“Unigram Language Model” segmentation

- SentencePiece incorporates Unigram as an **alternative to BPE**
- Vocabulary size still a hyperparameter, but the algorithm is **quite different**
  - Instead of growing the vocab by merging symbols, **shrink vocab by “pruning”**
  - Large **“seed vocabulary”** is formed from common character-sequences
  - The **overall frequency** of each item is calculated (**unigram probability**)
  - Probability of the data is calculated over **all possible segmentations** (as allowed by the current vocabulary)

“Unigram Language Model” segmentation

- SentencePiece incorporates Unigram as an **alternative to BPE**
- Vocabulary size still a hyperparameter, but the algorithm is **quite different**
  - Instead of growing the vocab by merging symbols, **shrink vocab by “pruning”**
  - Large **“seed vocabulary”** is formed from common character-sequences
  - The **overall frequency** of each item is calculated (**unigram probability**)
  - Probability of the data is calculated over **all possible segmentations** (as allowed by the current vocabulary)
  - At each step, **prune** the vocab item that **hurts text probability the least**

“Unigram Language Model” segmentation

- SentencePiece incorporates Unigram as an **alternative to BPE**
- Vocabulary size still a hyperparameter, but the algorithm is **quite different**
  - Instead of growing the vocab by merging symbols, **shrink vocab by “pruning”**
  - Large “**seed vocabulary**” is formed from common character-sequences
  - The **overall frequency** of each item is calculated (**unigram probability**)
  - Probability of the data is calculated over **all possible segmentations** (as allowed by the current vocabulary)
  - At each step, **prune** the vocab item that **hurts text probability the least**
  - Stop when the desired size is reached

Unigram vs. BPE

```
(base) ~/subword_demo --> more bpe.txt
#version: 0.2
t h
i n
a n
e r
o u
th e</w>
r e
in g</w>
a r
o n
e n
a l
an d</w>
```

```
(base) ~/subword_demo --> more unigram.vocab
<unk>   0
<s>     0
</s>    0
,       -3.0277
.       -3.14093
_the    -3.46301
s       -3.60884
_and    -4.05008
_a      -4.07705
_to     -4.14089
_of     -4.18806
'       -4.33785
ing     -4.5147
_I      -4.55371
```

Unigram vs. BPE

- Difference in **trained tokenization model**
  - BPE: **ordered** series of **merges**
  - Unigram: subword **vocabulary** with **probabilities**

```
(base) ~/subword_demo --> more bpe.txt
#version: 0.2
t h
i n
a n
e r
o u
th e</w>
r e
in g</w>
a r
o n
e n
a l
an d</w>
```

```
(base) ~/subword_demo --> more unigram.vocab
<unk>    0
<s>      0
</s>     0
,        -3.0277
.        -3.14093
_the     -3.46301
s        -3.60884
_and     -4.05008
_a       -4.07705
_to      -4.14089
_of      -4.18806
'        -4.33785
ing      -4.5147
_I       -4.55371
```

Unigram vs. BPE

- Difference in **trained tokenization model**
  - BPE: **ordered** series of **merges**
  - Unigram: subword **vocabulary** with **probabilities**
- BPE yields **only one segmentation**; Unigram can support **multiple**, but we usually pick the **most probable**

```
[(base) ~/subword_demo --> more bpe.txt
#version: 0.2
t h
i n
a n
e r
o u
th e</w>
r e
in g</w>
a r
o n
e n
a l
an d</w>
```

```
[(base) ~/subword_demo --> more unigram.vocab
<unk>   0
<s>     0
</s>    0
,       -3.0277
.       -3.14093
_the    -3.46301
s       -3.60884
_and    -4.05008
_a      -4.07705
_to     -4.14089
_of     -4.18806
'       -4.33785
ing     -4.5147
_I      -4.55371
```

Subword Regularization

Subwords (_ means spaces)	Vocabulary id sequence
_Hell/o/_world	13586 137 255
_H/ello/_world	320 7363 255
_He/llo/_world	579 10115 255
_/He/l/l/o/_world	7 18085 356 356 137 255
_H/el/l/o/_/world	320 585 356 137 7 12295

Table 1: Multiple subword sequences encoding the same sentence "Hello World"

Model	BLEU
Word	23.12
Character (512 nodes)	22.62
Mixed Word/Character	24.17
BPE	24.53
Unigram w/o SR ($l = 1$)	24.50
Unigram w/ SR ($l = 64,\ \alpha = 0.1$)	25.04

Table 5: Comparison of different segmentation algorithms (WMT14 en$\rightarrow$de)

source

Subword Regularization

- Sometimes beneficial to **train on multiple possible segmentations**
  - We might **not know the optimal segmentation** for a certain task

Subwords (_ means spaces)	Vocabulary id sequence
_Hell/o/_world	13586 137 255
_H/ello/_world	320 7363 255
_He/llo/_world	579 10115 255
_/He/l/l/o/_world	7 18085 356 356 137 255
_H/el/l/o/_/world	320 585 356 137 7 12295

Table 1: Multiple subword sequences encoding the same sentence "Hello World"

Model	BLEU
Word	23.12
Character (512 nodes)	22.62
Mixed Word/Character	24.17
BPE	24.53
Unigram w/o SR ($l = 1$)	24.50
Unigram w/ SR ($l = 64,\ \alpha = 0.1$)	25.04

Table 5: Comparison of different segmentation algorithms (WMT14 en$\rightarrow$de)

source

Subword Regularization

- Sometimes beneficial to **train on multiple possible segmentations**
  - We might **not know the optimal segmentation** for a certain task
- Subword Regularization: during training, **randomly sample** a possible segmentation when tokenizing
  - More **robust results for NMT**

Subwords (_ means spaces)	Vocabulary id sequence
_Hell/o/_world	13586 137 255
_H/ello/_world	320 7363 255
_He/llo/_world	579 10115 255
_/He/l/l/o/_world	7 18085 356 356 137 255
_H/el/l/o/_/world	320 585 356 137 7 12295

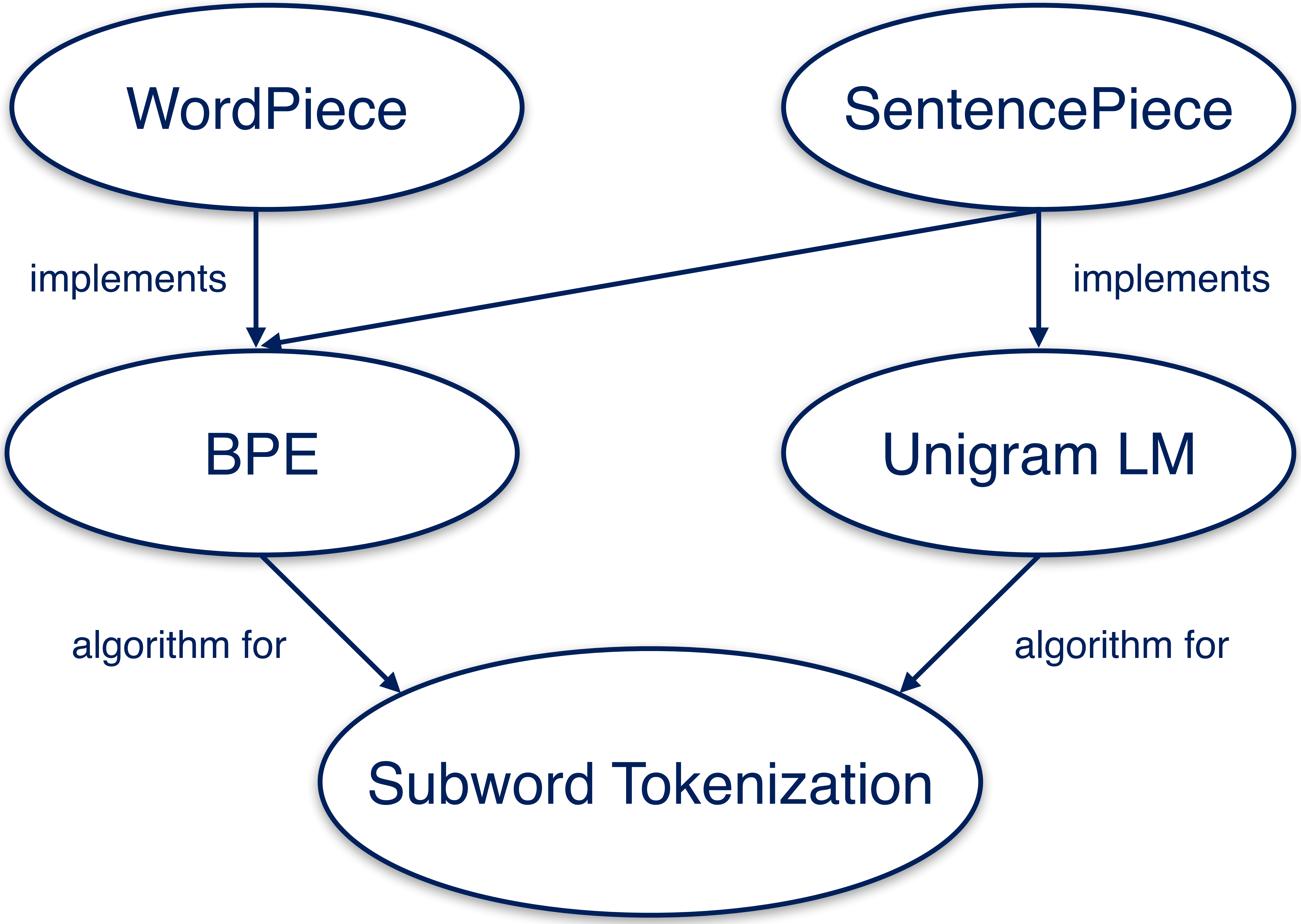
Table 1: Multiple subword sequences encoding the same sentence "Hello World"

Model	BLEU
Word	23.12
Character (512 nodes)	22.62
Mixed Word/Character	24.17
BPE	24.53
Unigram w/o SR ($l = 1$)	24.50
Unigram w/ SR ($l = 64,\ \alpha = 0.1$)	25.04

Table 5: Comparison of different segmentation algorithms (WMT14 en$\rightarrow$de)

source

Summary of Terms



Subwords vs. other segmentation

Morphological Segmentation

_the _n ation _sl ow ly _start ed _being _cent ral ized _and _d ur ing

source

Morphological Segmentation

- The results of BPE/Unigram tokenization does **not** tend to reflect **meaningful linguistic units (morphemes)**

_the _n ation _sl ow ly _start ed _being _cent ral ized _and _d ur ing

source

Morphological Segmentation

- The results of BPE/Unigram tokenization does **not** tend to reflect **meaningful linguistic units (morphemes)**
- In the example below: green = valid morpheme, blue = contiguous morphemes, red = not a morpheme / wrong use of morpheme

_the _n ation _sl ow ly _start ed _being _cent ral ized _and _d ur ing

source

Morphological Segmentation

- The results of BPE/Unigram tokenization does **not** tend to reflect **meaningful linguistic units (morphemes)**
- In the example below: green = valid morpheme, blue = contiguous morphemes, red = not a morpheme / wrong use of morpheme
- Extensive **disagreement** on whether morphological segmentation is more useful for tasks like **Machine Translation**
  - Overall: **not clear** that morphological segmentation helps

_the _n ation _sl ow ly _start ed _being _cent ral ized _and _d ur ing

source

Segmentation Research

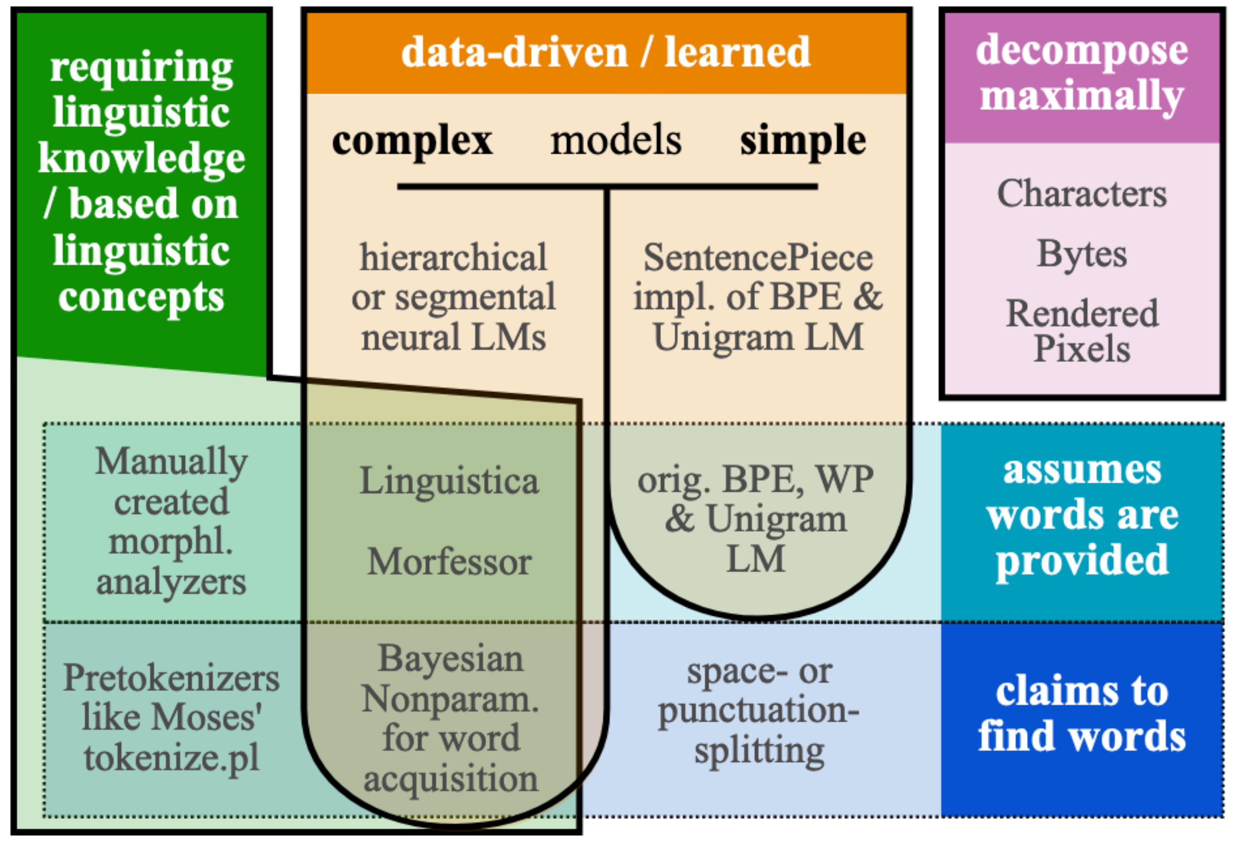
Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP

Sabrina J. Mielke [1,2] Zaid Alyafeai [3] Elizabeth Salesky [1]
Colin Raffel [2] Manan Dey [4] Matthias Gallé [5] Arun Raja [6]
Chenglei Si [7] Wilson Y. Lee [8] Benoît Sagot [9*] Samson Tan [10*]
BigScience Workshop Tokenization Working Group

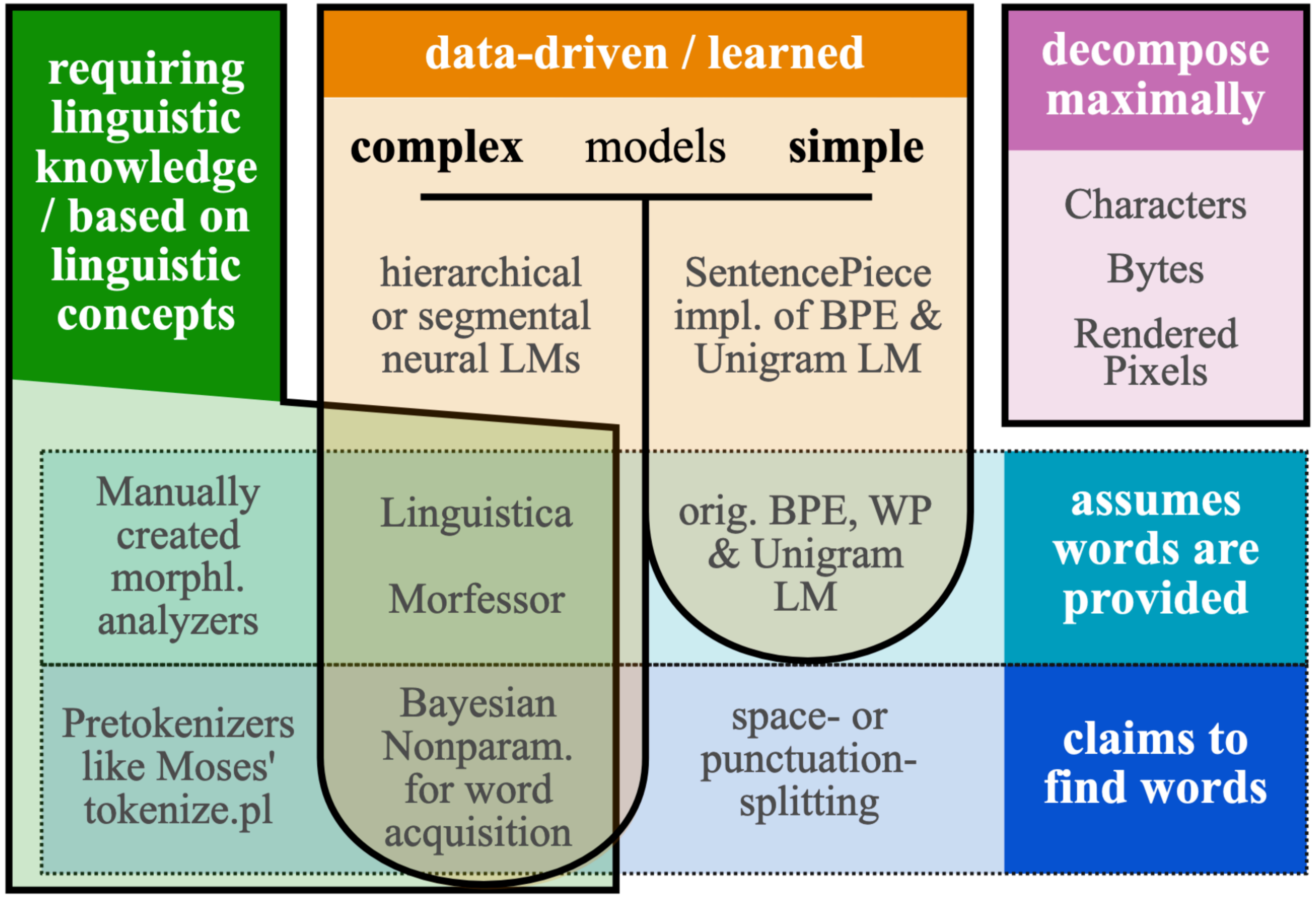
[1]Johns Hopkins University [2]HuggingFace [3]King Fahd University of Petroleum and Minerals [4]SAP
[5]Naver Labs Europe [6]Institute for Infocomm Research, A*STAR Singapore [7]University of Maryland
[8]BigScience Workshop [9]Inria Paris [10]Salesforce Research Asia & National University of Singapore
sjm@sjmielke.com

Abstract

What are the units of text that we want to model? From bytes to multi-word expressions, text can be analyzed and generated at many granularities. Until recently, most natural language processing (NLP) models operated over words, treating those as discrete and atomic tokens, but starting with byte-pair encoding (BPE), subword-based approaches have become dominant in many areas, enabling small vocabularies while still allowing for fast inference. Is the end of the road character-level model or byte-level pro-

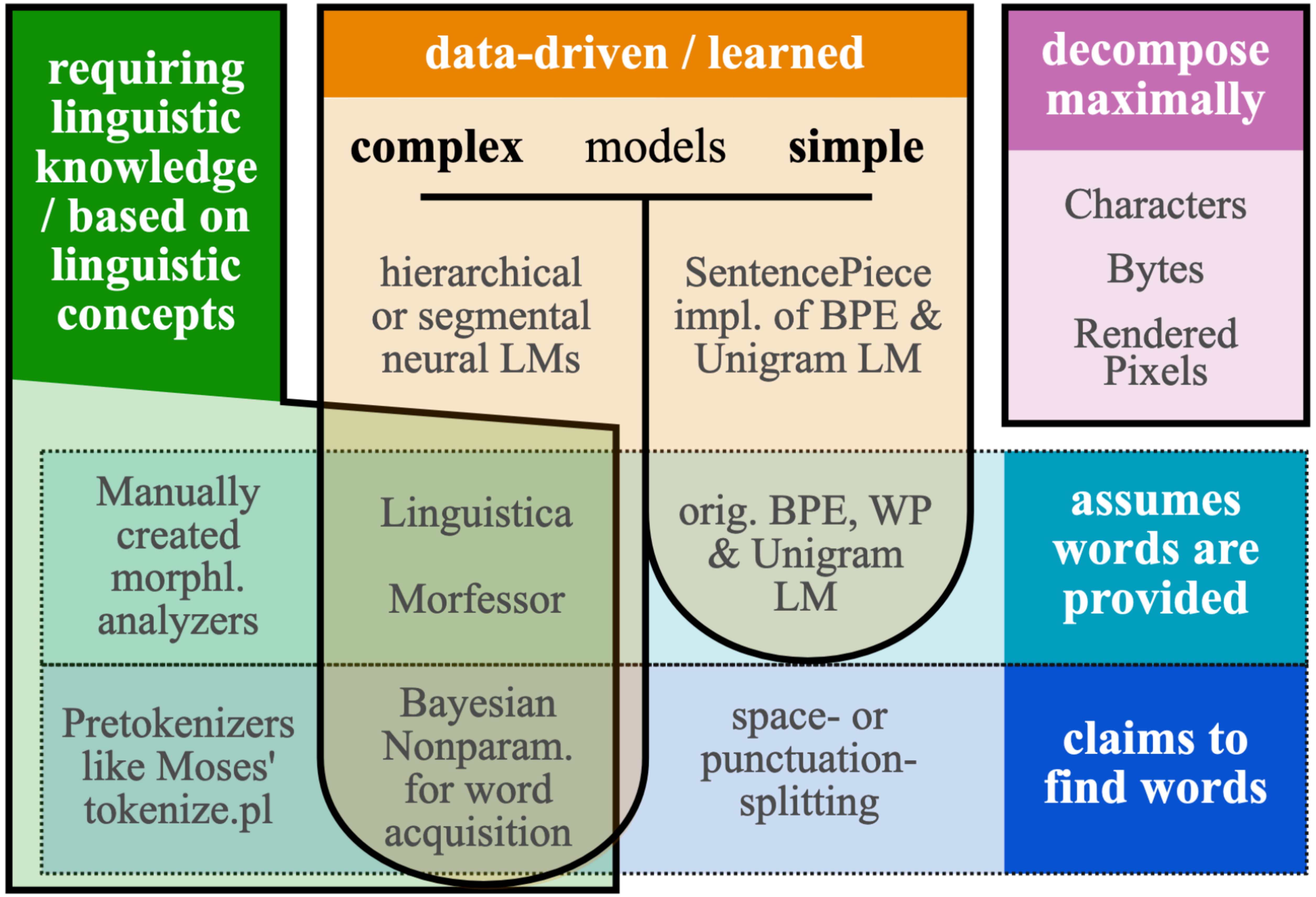


Why use subwords?



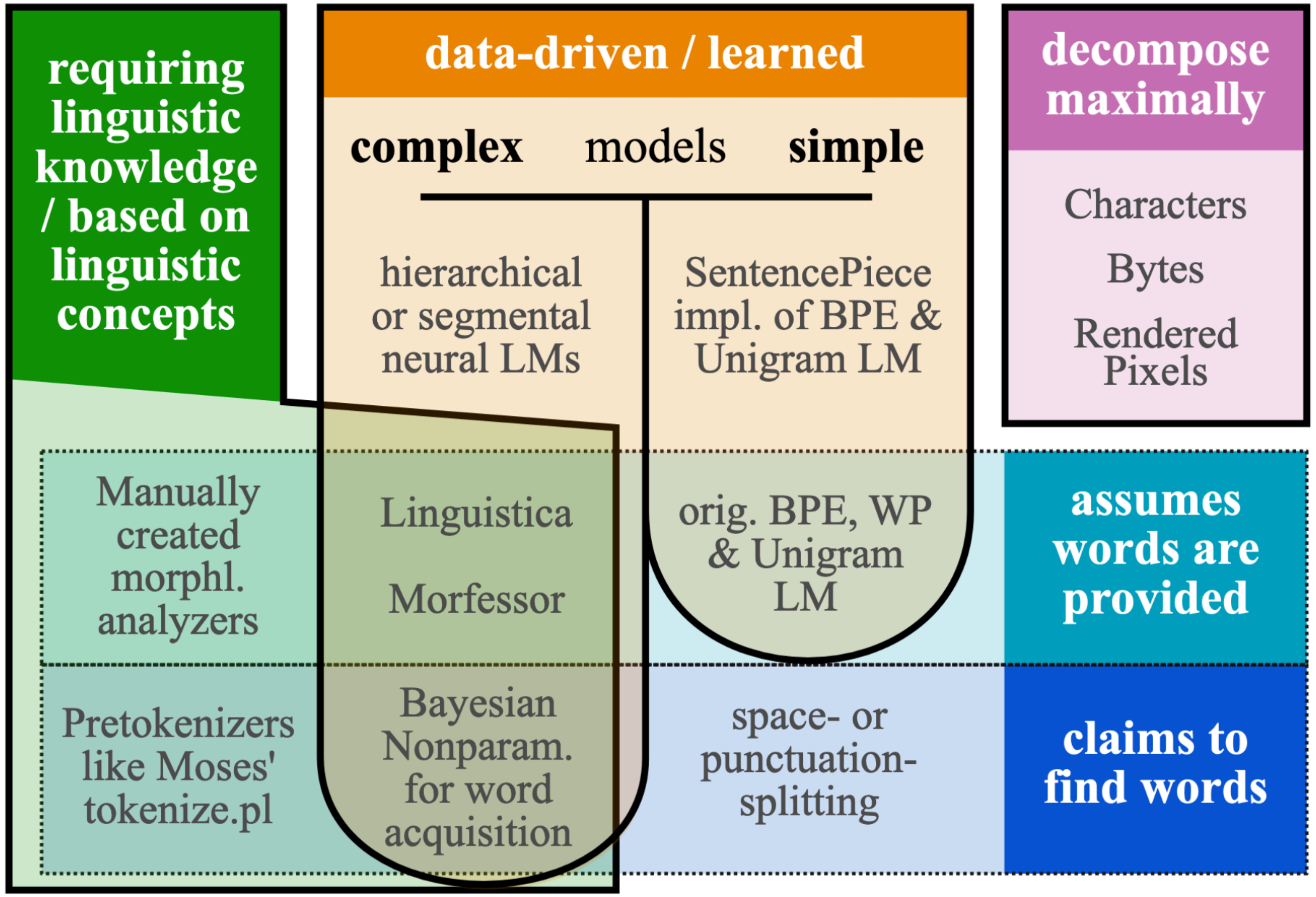
Why use subwords?

- Algorithms like SentencePiece are **simple** and **data-driven**
  - **Assume** very little
  - **Efficiently encode** the data
  - **Easy to train**



Why use subwords?

- Algorithms like SentencePiece are **simple** and **data-driven**
  - **Assume** very little
  - **Efficiently encode** the data
  - **Easy to train**
- Also demonstrate great **empirical performance**
  - Solve the OOV issue
  - Neural LMs seem to have no problem with subwords



Questions / Demonstration