BOSTON UNIVERSITY

Transformers – Part 2

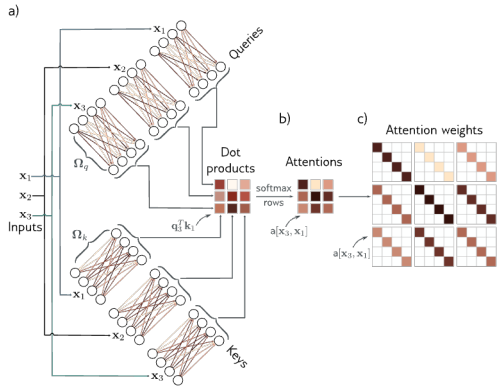
DL4DS – Spring 2026

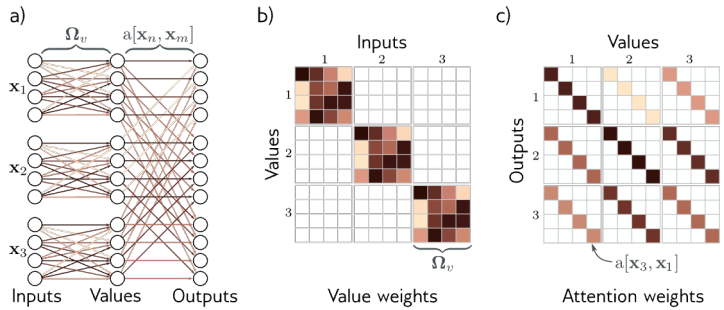
Today

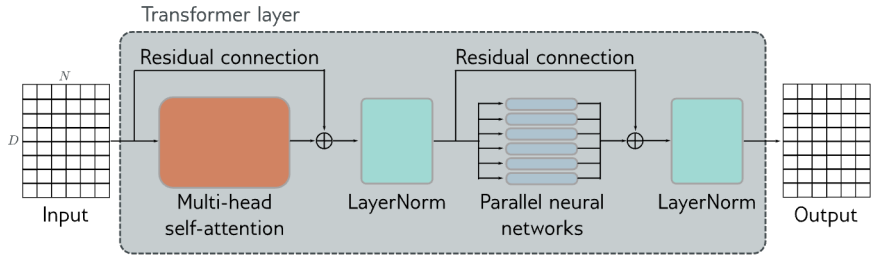
- Recap of Transformers Part 1
- Tokenization and Word Embedding
- Next token selection
- Transformers for Long Sequences

Recap From Part 1

- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models
  - Encoder
  - Decoder
  - Encoder-Decoder







3 Types of Transformer Models

1. *Encoder* – transforms text embeddings into representations that support variety of tasks (e.g. sentiment analysis, classification)
   - ❖ Model Example: BERT

2. *Decoder* – predicts the next token to continue the input text (e.g. ChatGPT, AI assistants)
   - ❖ Model Example: GPT4, GPT4, …

3. *Encoder-Decoder* – used in sequence-to-sequence tasks, where one text string is converted to another (e.g. machine translation)

Tokenization and Word Embedding

What's a Token?

A small chunk of text that we use to aid language modeling.

- Represents one or more bytes
- Input texts are greedily divided into tokens.
  - Longest prefix matching a token.
- Token set also constructed greedily.
  - Start with 256 possible bytes.
  - Then greedily pick the most common pairs of adjacent tokens.

Why Tokens?

Instead of…

- Bits - not enough semantics* and missing intrabyte positioning
- Bytes - not enough semantics* for Unicode
- Characters - too many of them if we try to support all languages
- Words - even more words than characters

Remember:

* One-hot/Softmax tactic means we will have at least one output per possible output value, and many more parameters in practice.

Unicode Standard and UTF-8

- Unicode – *variable length* character encoding standard. currently defines 149,813 characters and 161 scripts, including emoji, symbols, etc.
- Unicode Codepoint – can represent up to $17 \times 2^{16} = 1{,}114{,}112$ entries. e.g. U+0000 – U+10FFFF in hexadecimal
- Unicode Transformation Standard (e.g. UTF-8) – is a *variable length encoding* using one to four bytes
  - First 128 chars same as ASCII

Code point ↔ UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0xxxxxxx			
U+0080	U+07FF	110xxxxx	10xxxxxx		
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+010000	[b]U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

128 code points, 1 byte. Covers ASCII

1920 code points, 2 bytes. Covers remainder of almost all Latin-script alphabets all Latin-script alphabets, and also IPA extensions, Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac, Thaana and N'Ko alphabets, as well as Combining Diacritical Marks

61,440 code points, 3 bytes. Basic Multilingual Plane including Chinese, Japanese and Korean characters

1,048,567 code points 4 bytes. Emoji, historic scripts, math symbols

https://en.wikipedia.org/wiki/Unicode
https://en.wikipedia.org/wiki/UTF-8

ASCII – American Standard Code for Information Interchange

Bits $b_7 b_6 b_5$ →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	Row ↓ Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

https://en.wikipedia.org/wiki/ASCII
https://en.wikipedia.org/wiki/ASCII#/media/File:USASCII_code_chart.svg

Example Tokens

```
import tiktoken
```

```
[6] enc = tiktoken.encoding_for_model("gpt-4o")
```

```
for i in range(1024):
    d = enc.decode([i])
    if len(d) >= 4:
        print(i, enc.decode([i]))
```

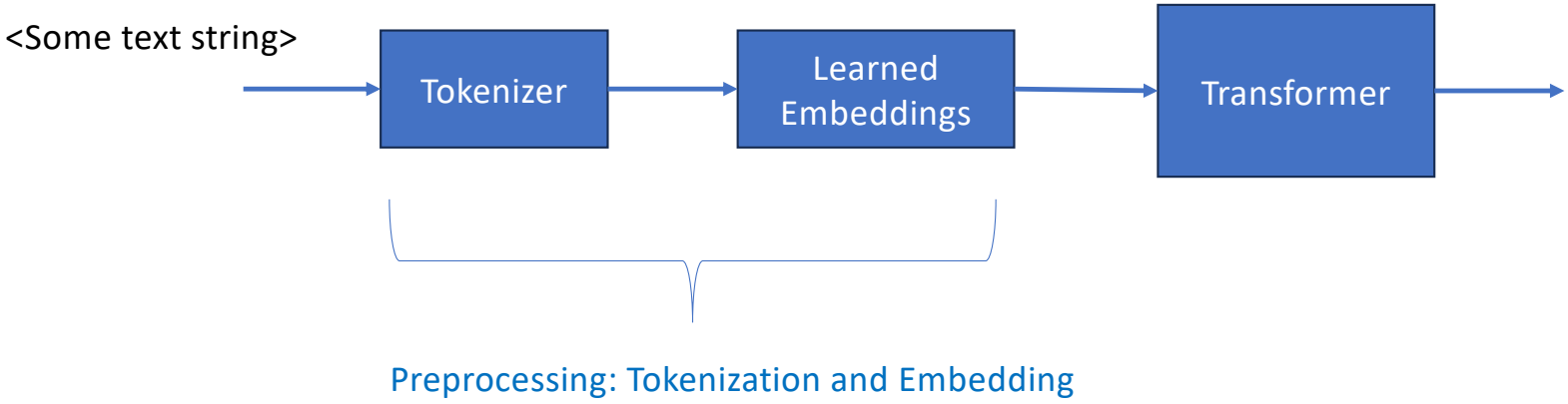
```
257
269
290  the
309
326  and
352
387 ation
395  for
406  con
408
440  pro
447 port
452  com
475 ction
481  you
483  with
484  that
495  this
506
508 ment
518 ----
529 turn
530
553  are
561 import
562 able
583 ight
584 ublic
591  from
595 ****
600 tring
620  new
622  return
623  The
625  not
626
634  your
661  que
665  can
673  was
677  int
679  have
686  par
694  res
699
700 form
717  get
722  all
728 ject
731  des
735 alue
738  will
740 ();
744  class
751  public
756 ions
758  }
763 --------
766 ance
767 ould
773 ient
775 .get
```

https://github.com/openai/tiktoken

NLP Preprocessing Pipeline

Transformers don't work on character string directly, but rather on vectors.

The character strings must be converted to vectors



Tokenizer

Tokenizer chooses input "units", e.g. words, sub-words, characters via *tokenizer training*

In tokenizer training, commonly occurring substrings are greedily merged based on their frequency, starting with character pairs

Tokenization Issues

“A lot of the issues that may look like issues with the neural network architecture actually trace back to tokenization. Here are just a few examples” – Andrej Karpathy

- Why can't LLM spell words? Tokenization.
- Why can't LLM do super simple string processing tasks like reversing a string? Tokenization.
- Why is LLM worse at non-English languages (e.g. Japanese)? Tokenization.
- Why is LLM bad at simple arithmetic? Tokenization.
- Why did GPT-2 have more than necessary trouble coding in Python? Tokenization.
- Why did my LLM abruptly halt when it sees the string "<|endoftext|>"? Tokenization.
- What is this weird warning I get about a "trailing whitespace"? Tokenization.
- Why did the LLM break if I ask it about "SolidGoldMagikarp"? Tokenization.
- Why should I prefer to use YAML over JSON with LLMs? Tokenization.
- Why is LLM not actually end-to-end language modeling? Tokenization.
- What is the real root of suffering? Tokenization.

https://github.com/karpathy/minbpe/blob/master/lecture.md

Tokenization Matters

From the gpt-4o announcement,

“It matches GPT-4 Turbo performance on text in English and code, with significant improvement on text in non-English languages, while also being much faster and 50% cheaper in the API.”

Gains were from increasing the number of tokens in the updated tokenizer.

https://openai.com/index/hello-gpt-4o/

Russian 1.7x fewer tokens (from 39 to 23)	Привет, меня зовут GPT-4o. Я — новая языковая модель, приятно познакомиться!
Korean 1.7x fewer tokens (from 45 to 27)	안녕하세요, 제 이름은 GPT-4o입니다. 저는 새로운 유형의 언어 모델입니다, 만나서 반갑습니다!
Vietnamese 1.5x fewer tokens (from 46 to 30)	Xin chào, tên tôi là GPT-4o. Tôi là một loại mô hình ngôn ngữ mới, rất vui được gặp bạn!
Chinese 1.4x fewer tokens (from 34 to 24)	你好，我的名字是GPT-4o。我是一种新型的语言模型，很高兴见到你!
Japanese 1.4x fewer tokens (from 37 to 26)	こんにちは、私の名前はGPT-4oです。私は新しいタイプの言語モデルです。初めまして！

Unicode Standard and UTF-8

- Unicode – *variable length* character encoding standard. currently defines 149,813 characters and 161 scripts, including emoji, symbols, etc.
- Unicode Codepoint – can represent up to $17 \times 2^{16} = 1{,}114{,}112$ entries. e.g. U+0000 – U+10FFFF in hexadecimal
- Unicode Transformation Standard (e.g. UTF-8) – is a *variable length encoding* using one to four bytes
  - First 128 chars same as ASCII

Code point ↔ UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4	
U+0000	U+007F	0xxxxxxx				Covers ASCII
U+0080	U+07FF	110xxxxx	10xxxxxx			Covers remainder of almost all Latin-script alphabets
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx		Basic Multilingual Plane including Chinese, Japanese and Korean characters
U+010000	[b]U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx	Emoji, historic scripts, math symbols

https://en.wikipedia.org/wiki/Unicode
https://en.wikipedia.org/wiki/UTF-8

Tokenizer

Two common tokenizers:

- Byte Pair Encoding (BPE) – Used by OpenAI GPT2, GPT4, etc.
  - The BPE algorithm is "byte-level" because it runs on UTF-8 encoded strings.
  - This algorithm was popularized for LLMs by the GPT-2 paper and the associated GPT-2 code release from OpenAI. Sennrich et al. 2015 is cited as the original reference for the use of BPE in NLP applications. Today, all modern LLMs (e.g. GPT, Llama, Mistral) use this algorithm to train their tokenizers.*
- sentencepiece
  - (e.g. Llama, Mistral) use sentencepiece instead. Primary difference being that sentencepiece runs BPE directly on Unicode code points instead of on UTF-8 encoded bytes.

* https://github.com/karpathy/minbpe/tree/master

BPE Pseudocode

```
Initialize vocabulary with individual characters in the text and their frequencies
While desired vocabulary size not reached:
    Identify the most frequent pair of adjacent tokens/characters in the vocabulary
    Merge this pair to form a new token
    Update the vocabulary with this new token
    Recalculate frequencies of all tokens including the new token
Return the final vocabulary
```

Enforce a Token Split Pattern

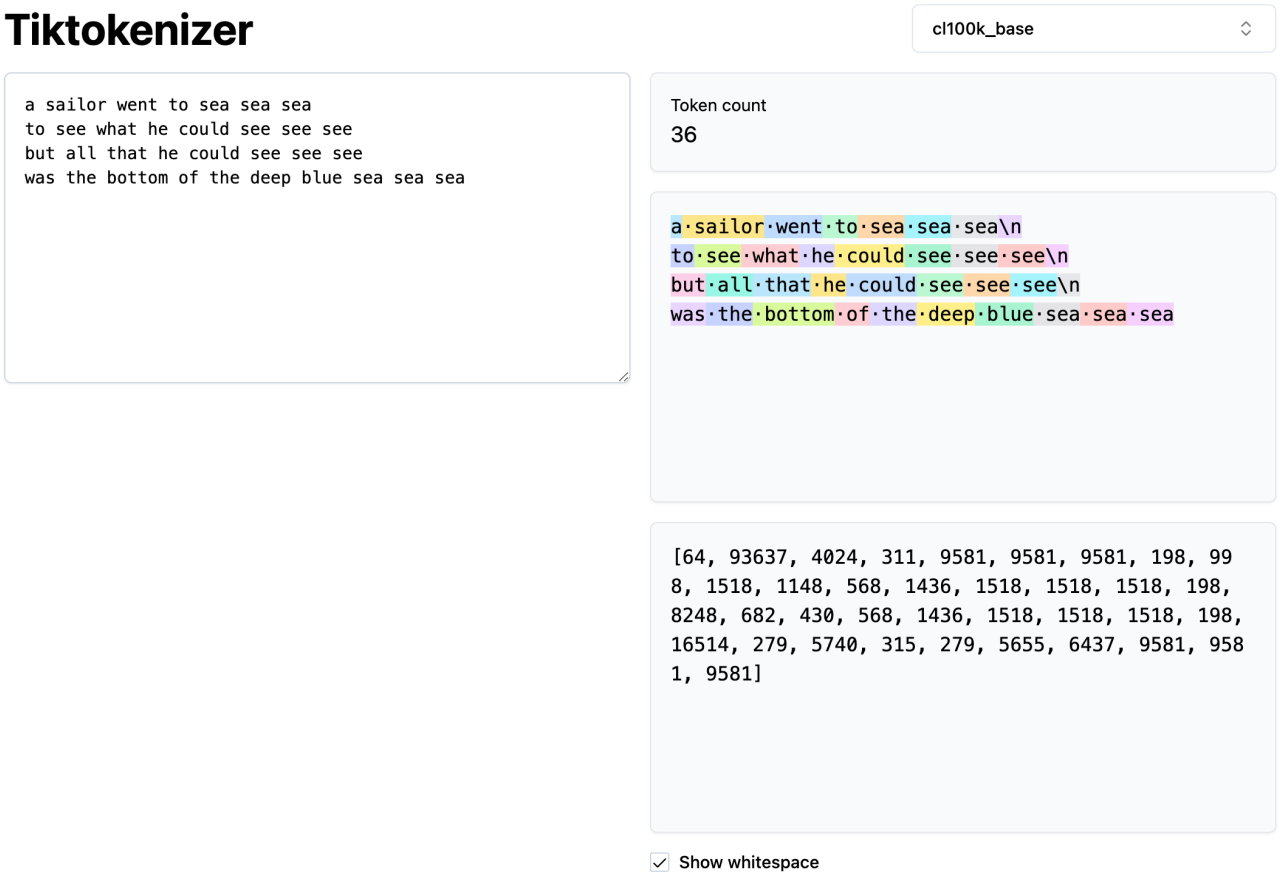
```
GPT2_SPLIT_PATTERN = r"""'(?:[sdmt]|ll|ve|re)| ?\p{L}+| ?\p{N}+|
?[^\s\p{L}\p{N}]+|\s+(?!\S)|\s+"""

GPT4_SPLIT_PATTERN = r"""'(?i:[sdmt]|ll|ve|re)|[^\r\n\p{L}\p{N}]?+\p{L}+|\p{N}{1,3}|
?[^\s\p{L}\p{N}]++[\r\n]*|\s*[\r\n]|\s+(?!\S)|\s+"""
```

- Do not allow tokens to merge across certain characters or patterns
- Common contraction endings: ‘ll, ‘ve, ‘re
- Match words with a leading space
- Match numeric sequences
- carriage returns, new lines

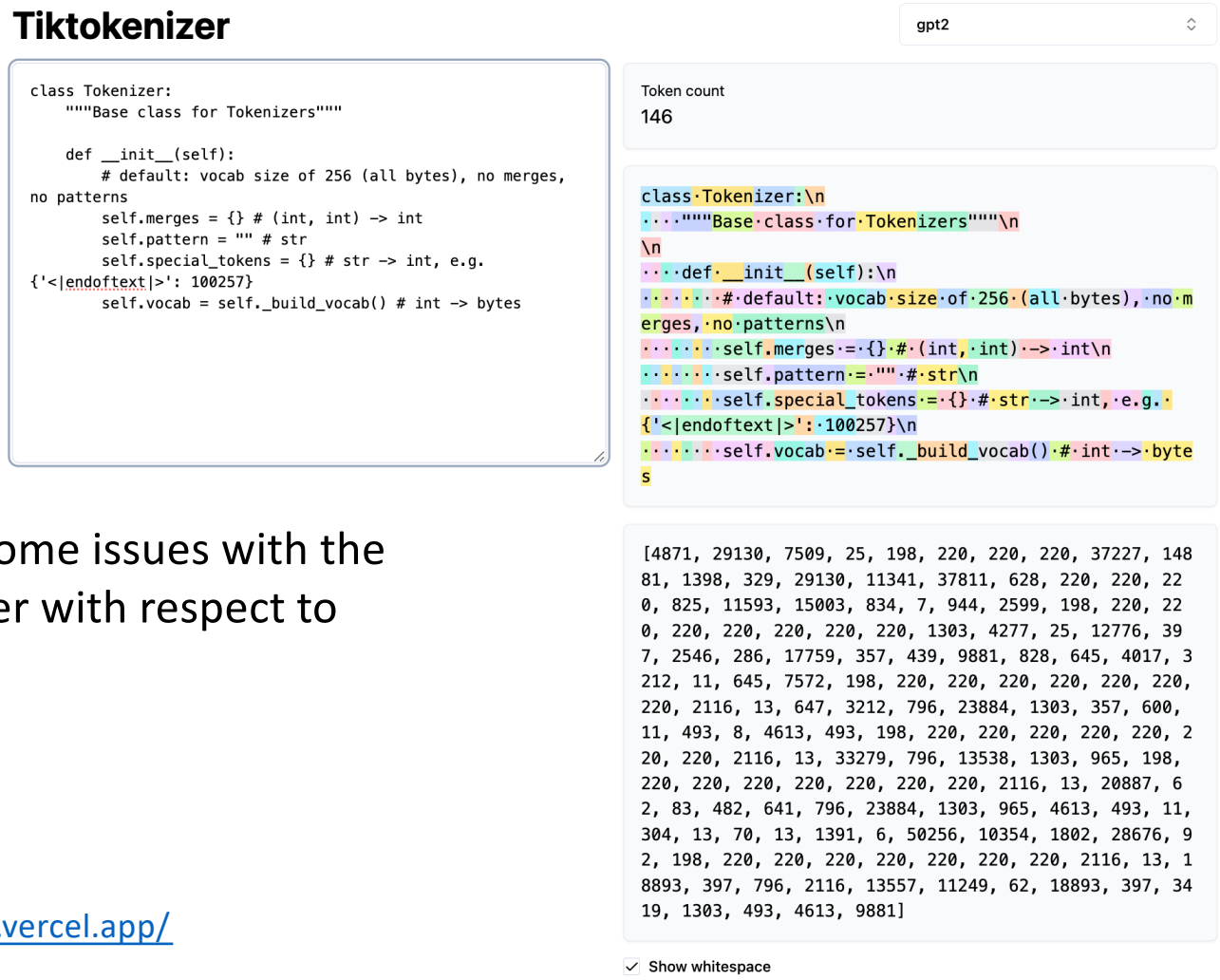
GPT4 Tokenizer

cl100k_base is the GPT4 tokenizer



https://tiktokenizer.vercel.app/

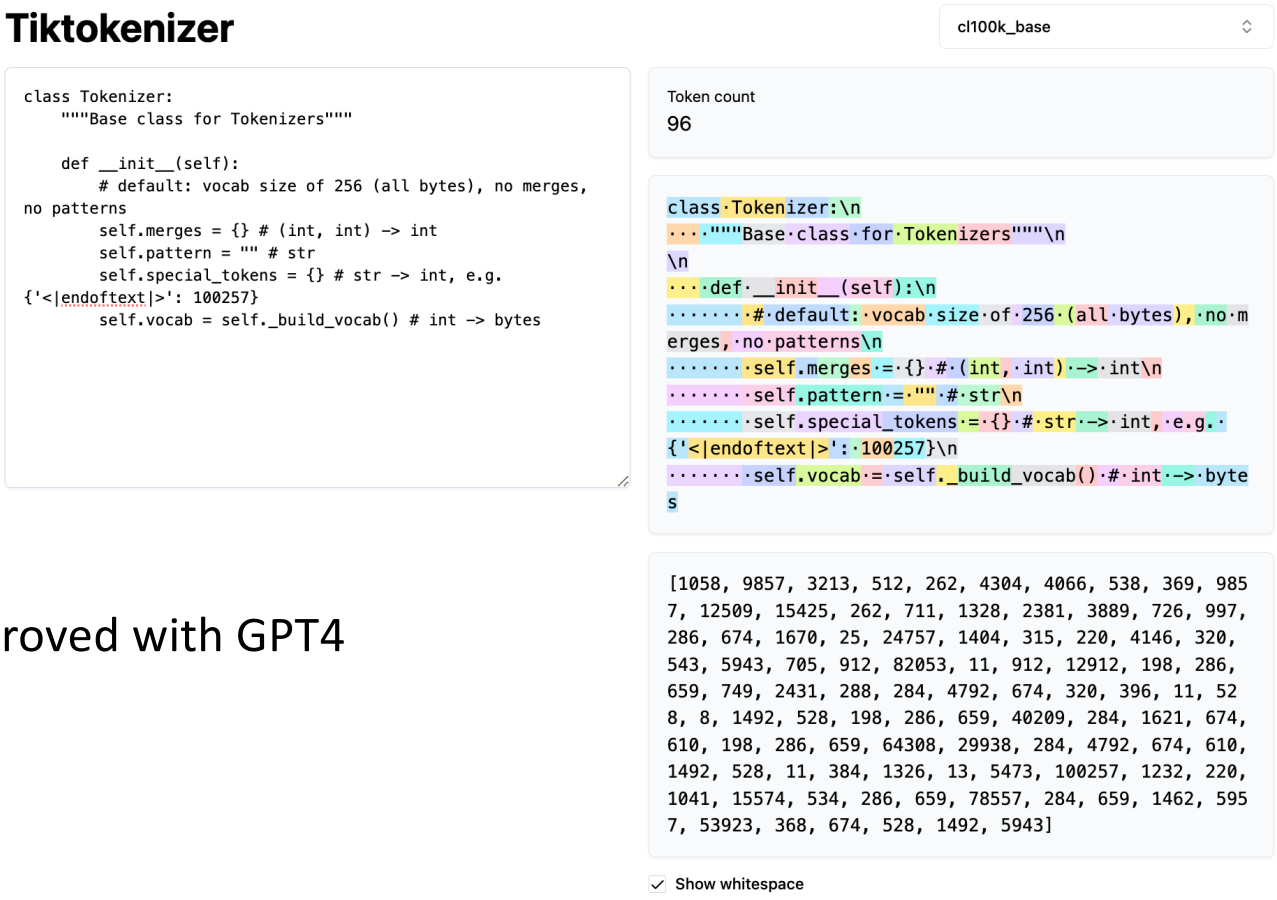
GPT2 Tokenizer



You can see some issues with the GPT2 tokenizer with respect to python code

https://tiktokenizer.vercel.app/

GPT4 Tokenizer



Issues are improved with GPT4 tokenizer

https://tiktokenizer.vercel.app/

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

Byte Pair Encoding (BPE) Example

Minimal starting vocabulary of subset of lower case latin alphabet and space `_`.

Byte Pair Encoding (BPE) Example

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

b)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

Find the most frequent pair of adjacent tokens, `se`, in this case and form new token.

Byte Pair Encoding (BPE) Example

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

b)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

c)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	1

Next most frequent pair of tokens is `e_`

Byte Pair Encoding (BPE) Example

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

b)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

c)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	1

⋮ ⋮

Continue until you hit your vocabulary size limit.

d)

see_	sea_	e	b	l	w	a	could_	hat_	he_	o	t	t_	the_	to_	u	a_	d	f	m	n	p	s	sailor_	to
7	6	4	3	3	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1

Byte Pair Encoding (BPE) Example

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

b)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

c)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	1

⋮ ⋮

d)

see_	sea_	e	b	l	w	a	could_	hat_	he_	o	t	t_	the_	to_	u	a_	d	f	m	n	p	s	sailor_	to
7	6	4	3	3	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1

⋮ ⋮ ⋮

e)

see_	sea_	could_	he_	the_	a_	all_	blue_	bottom_	but_	deep_	of_	sailor_	that_	to_	was_	went_	what_
7	6	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

b)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

c)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	1

⋮ ⋮

d)

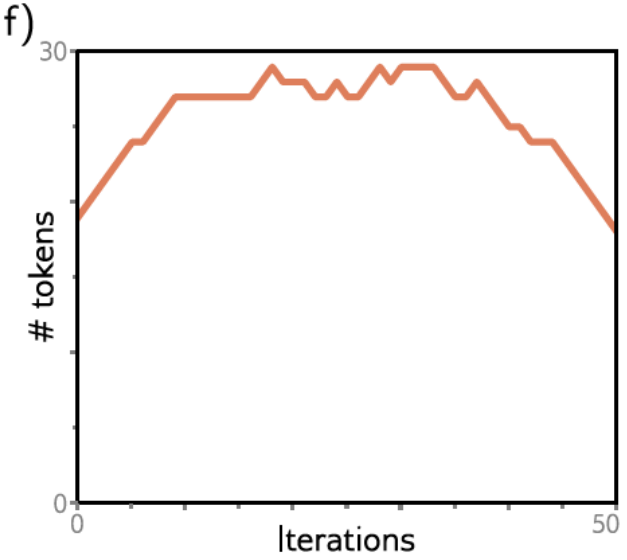
see_	sea_	e	b	l	w	a	could_	hat_	he_	o	t	t_	the_	to_	u	a_	d	f	m	n	p	s	sailor_	to
7	6	4	3	3	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1

⋮ ⋮ ⋮

e)

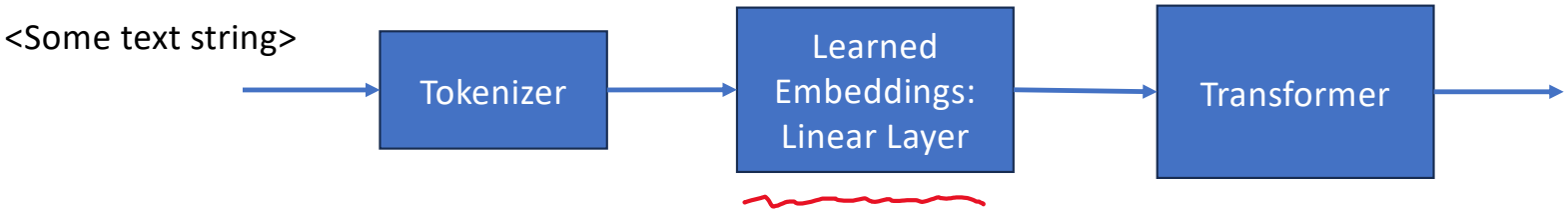
see_	sea_	could_	he_	the_	a_	all_	blue_	bottom_	but_	deep_	of_	sailor_	that_	to_	was_	went_	what_
7	6	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1

f)



Generally # of tokens increases and then starts decreasing after continuing to merge tokens

Learned Embeddings



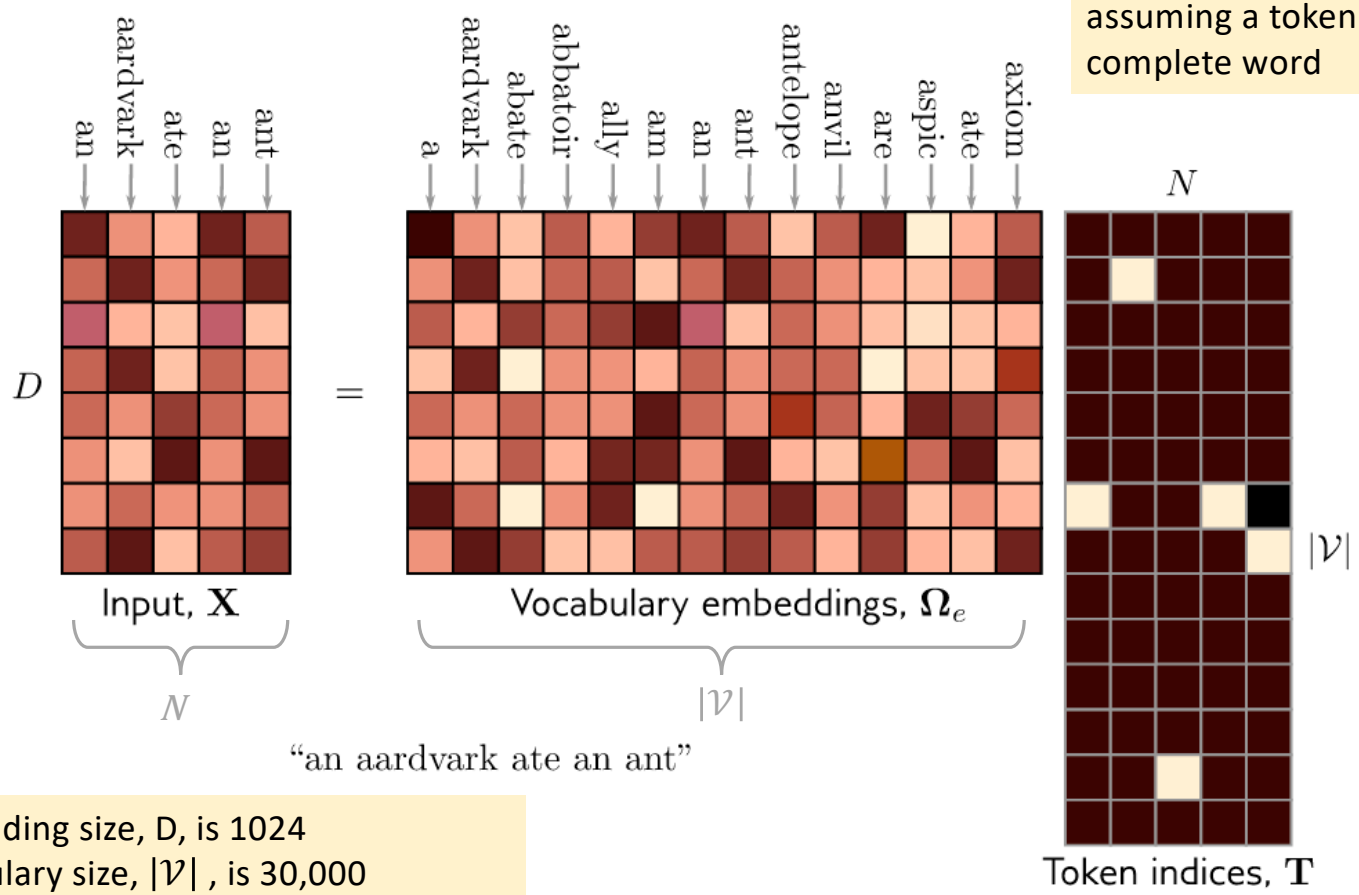
- After the tokenizer, you have an updated ”vocabulary” indexed by token ID
- Next step is to translate the token into an embedding vector
- Translation is done via a linear layer which is typically learned with the rest of the transformer model

```
self.embedding = nn.Embedding(vocab_size, embedding_dim)
```

- Special layer definition, likely to exploit sparsity of input

Embeddings Output

In this example, we are assuming a token is simply a complete word

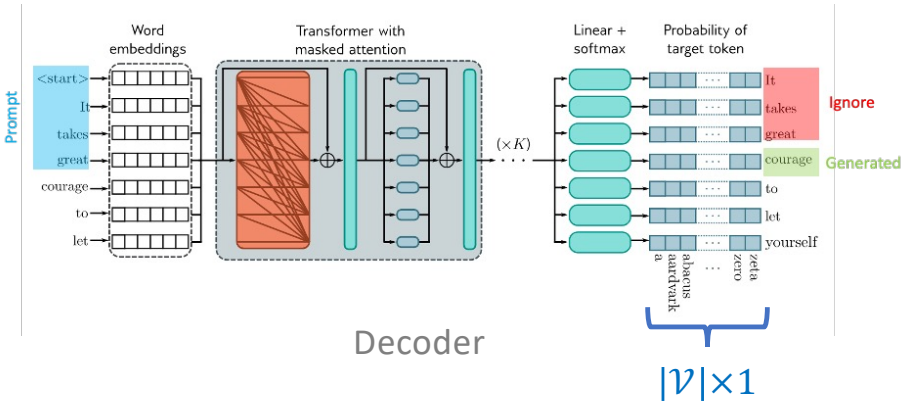


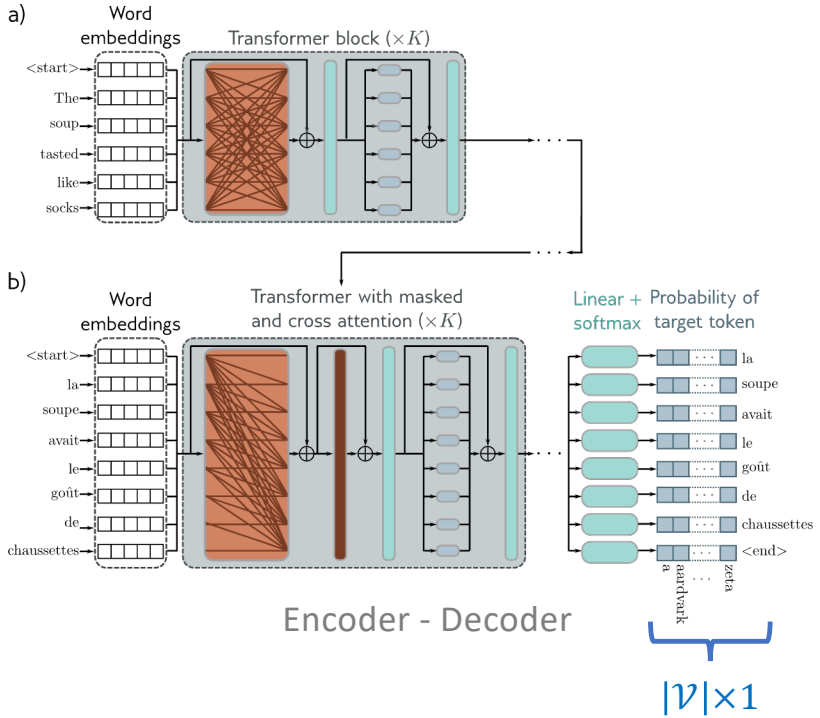
"One hot encoding"

- Typical embedding size, D, is 1024
- Typical vocabulary size, $|\mathcal{V}|$, is 30,000
- So 30M parameters just for this matrix!

Next Token Selection

Next Token Selection





- Recall: output is a $|\mathcal{V}|\times 1$ vector of probabilities
- How should we pick the next token?
- Trade off between accuracy and diversity

Next Token Selection

Recall: output is a $|\mathcal{V}|\times 1$ vector of probabilities

Selection methods:

- Greedy selection
- Top-K
- Nucleus
- Beam search

Next Token Selection – Greedy

Pick most likely token (greedy)

Simple to implement. Just take the max().

$$\hat{y}_t = \underset{w \in \mathcal{V}}{\operatorname{argmax}} \left[Pr(y_t = w | \hat{\mathbf{y}}_{<t}, \mathbf{x}, \boldsymbol{\phi}) \right]$$

```
# in PyTorch
outputs = model(inputs)
value, index = outputs.max(1)
```

Might pick first token y_0, but then there is no y_1 where $\Pr(y_1 | y_0)$ is high.

Result is generic and predictable. Same output for a given input context.

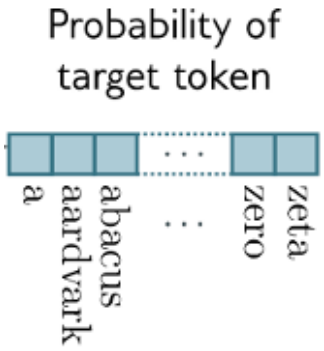
Next Token Selection -- Sampling

Sample from the probability distribution

Get a bit more diversity in the output

Will occasionally sample from the long tail of the distribution, producing some unlikely word combinations

Next Token Selection – Top *K* Sampling

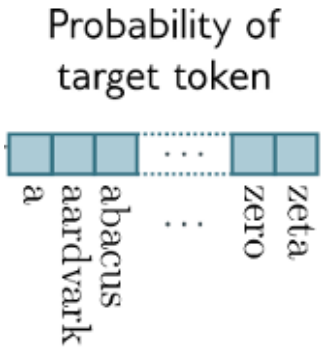


1. Generate the probability vector as usual
2. Sort tokens by likelihood
3. Discard all but top *k* most probable words
4. Renormalize the probabilities to be valid probability distribution (e.g. sum to 1)
5. Sample from the new distribution

Diversifies word selection

Depends on the distribution. Could be low variance, reducing diversity

Next Token Selection – Nucleus Sampling



Instead of keeping top-*k*, keep the top *p* percent of the probability mass.

Choose from the smallest set from the vocabulary such that

$$\sum_{w \in V^{(p)}} P(w|\mathbf{w}_{<t}) \geq p.$$

Diversifies word selection with less dependence on nature of distribution.

Depends on the distribution. Could be low variance, reducing diversity

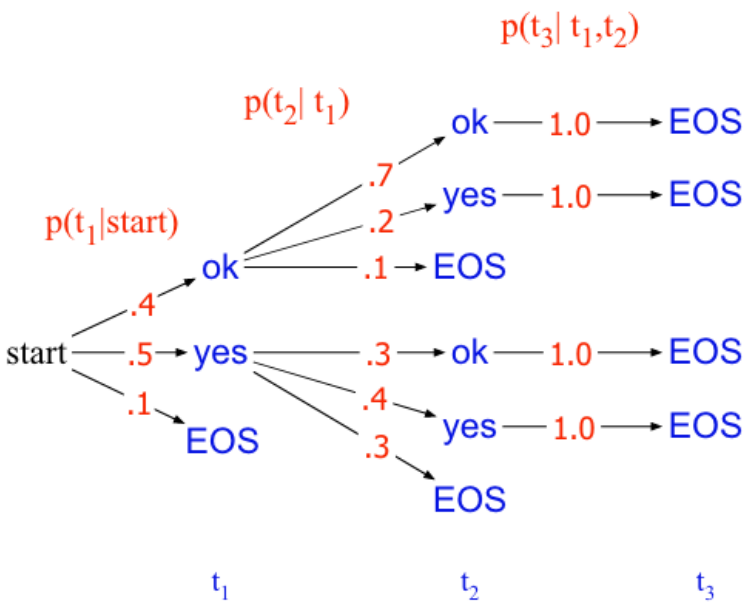
Next Token Selection – Beam Search

Commonly used in *machine translation*

Maintain multiple output choices and then choose best combinations later via tree search

V = {yes, ok, <eos>}

We want to maximize $p(t_1, t_2, t_3)$.



Greedy: $0.5 \times 0.4 \times 1.0 = 0.20$

Optimal: $0.4 \times 0.7 \times 1.0 = 0.28$

D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search

But we can’t exhaustively search the entire vocabulary

Keep k tokens (beam width) at each step

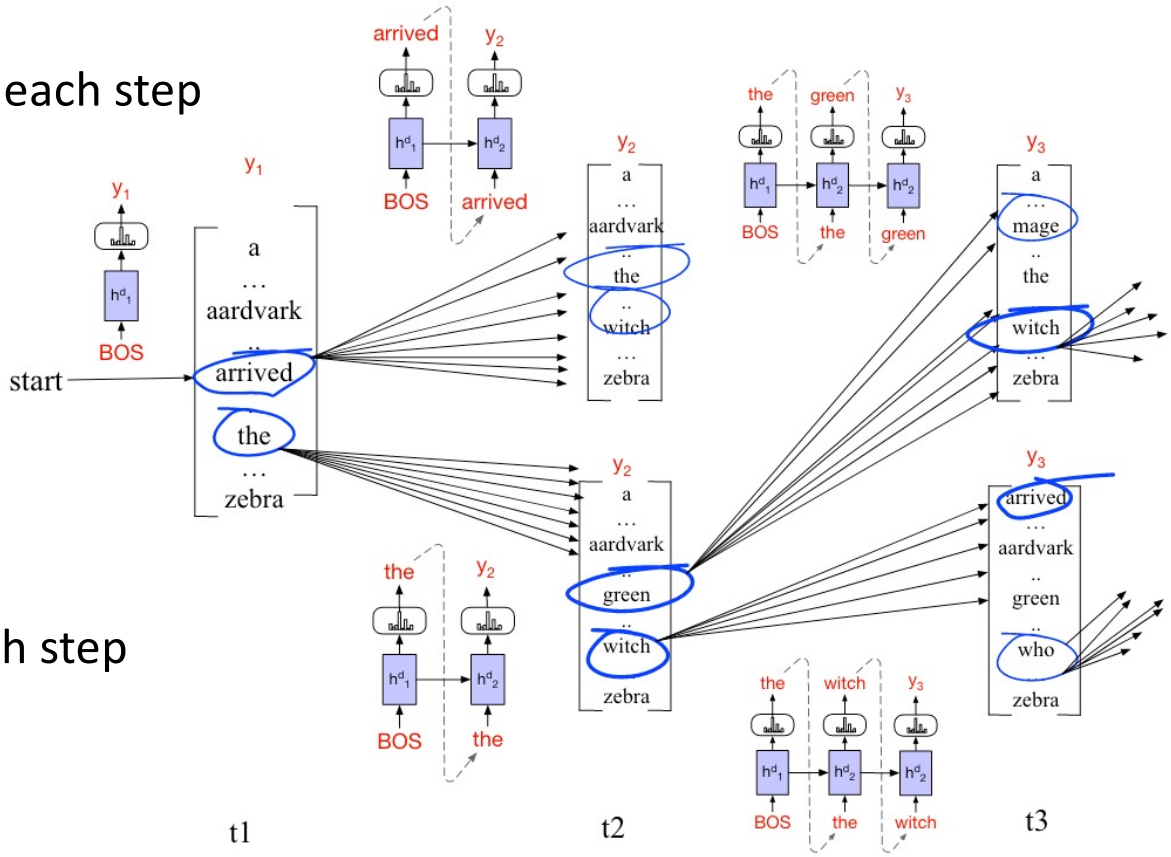
D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search

BOS: Beginning of Sentence token

Keep *k* tokens at each step

E.g. *k* = 2



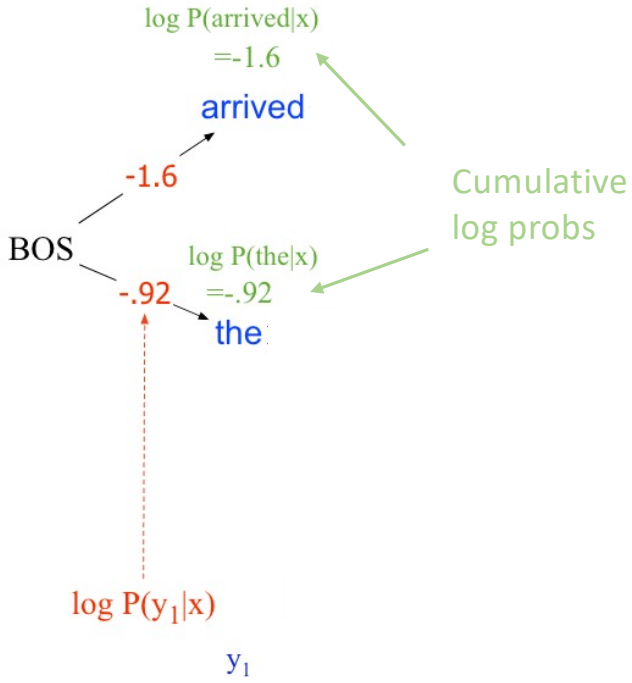
Prune to *k* at each step

D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search (k=2)

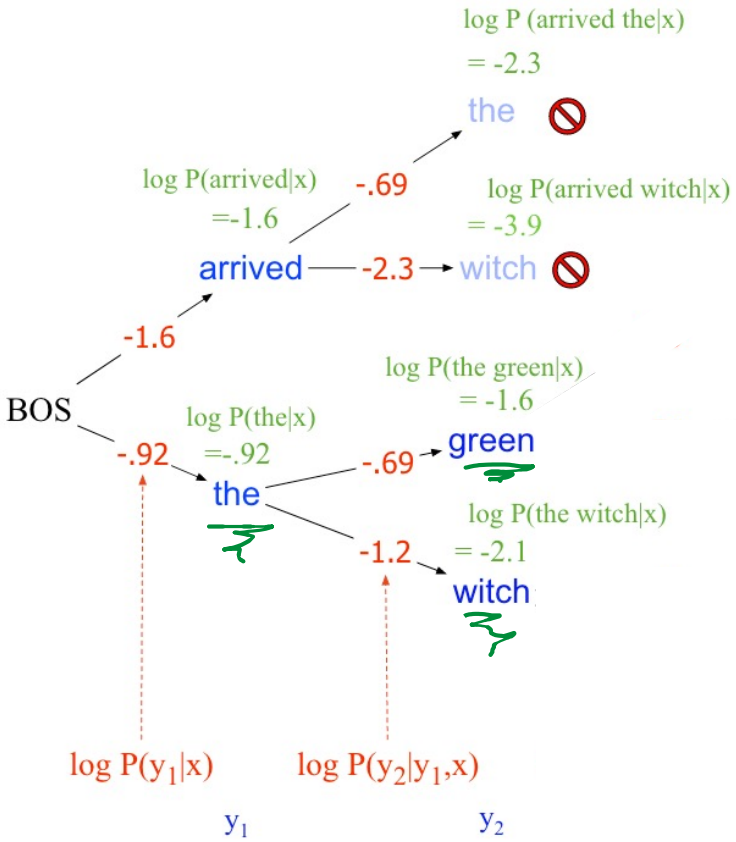
Calculated with *log probabilities* and add

Pick the top 2 tokens.



D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search (k=2)



Then pick the next 2 from each of the first 2 tokens.

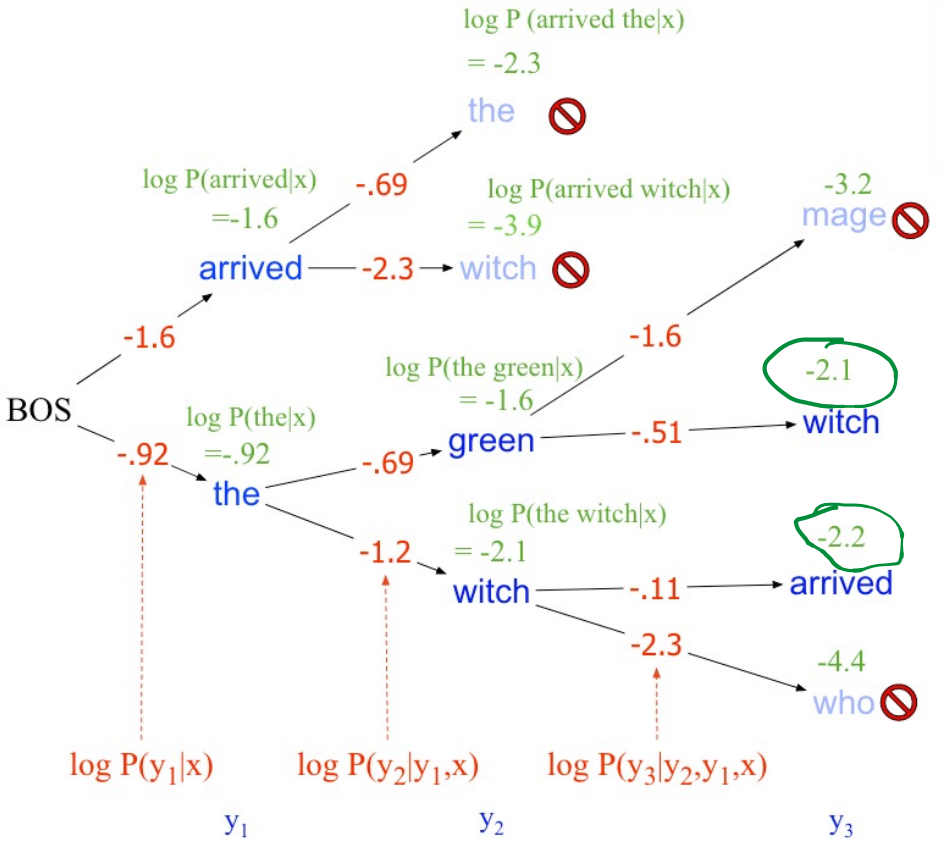
Calc cumulative log probs:
-1.6 -.69 = -2.3
-1.6 -2.3 = -3.9

-.92 -.69 = -1.6
-.92 -1.2 = -2.1

Pick the 1st token with highest log probability.

D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search (k=2)



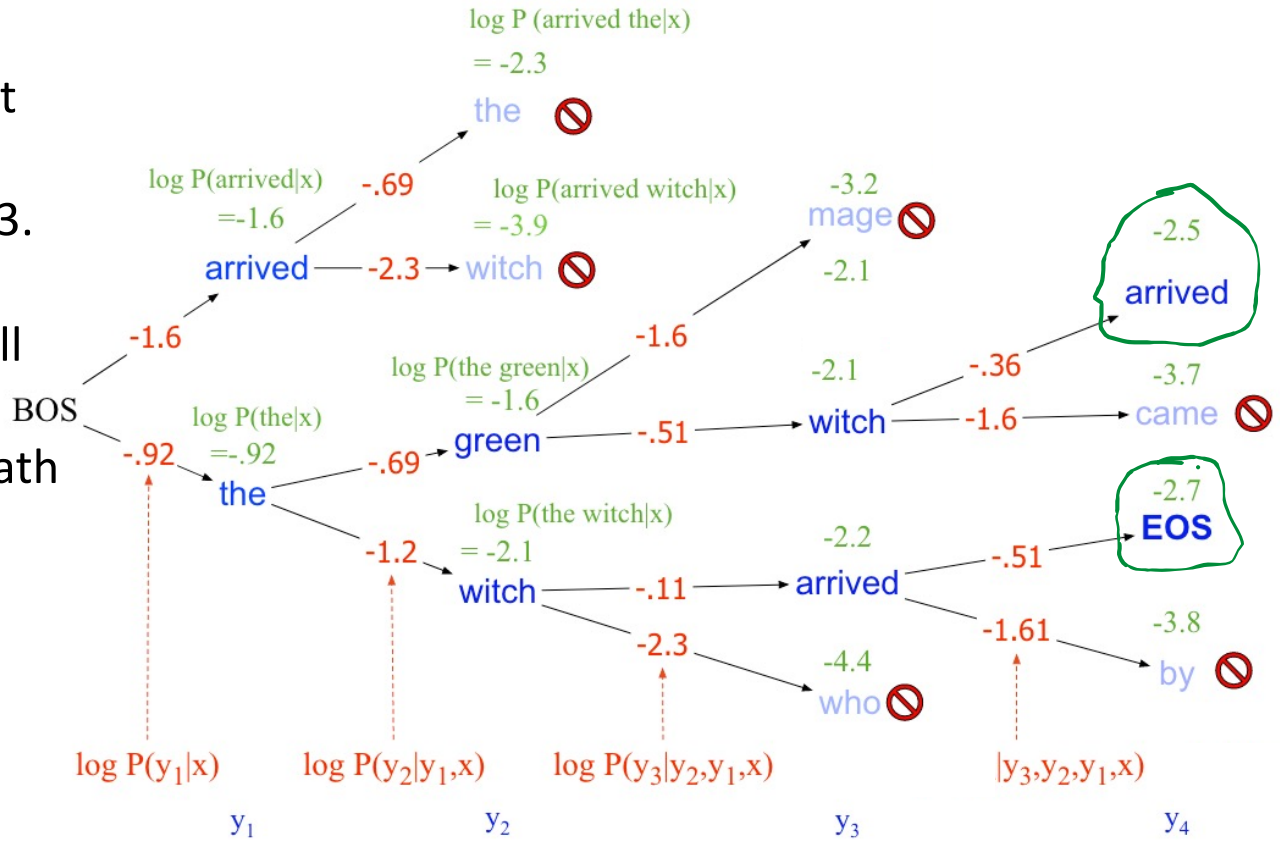
Then generate the next 2 tokens from each of the y2 and pick the 2 highest log probability paths.

D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search (k=2)

Continue to predict the next 2 highest from each of the y3.

We hit EOS, but still have a lower cumulative prob path

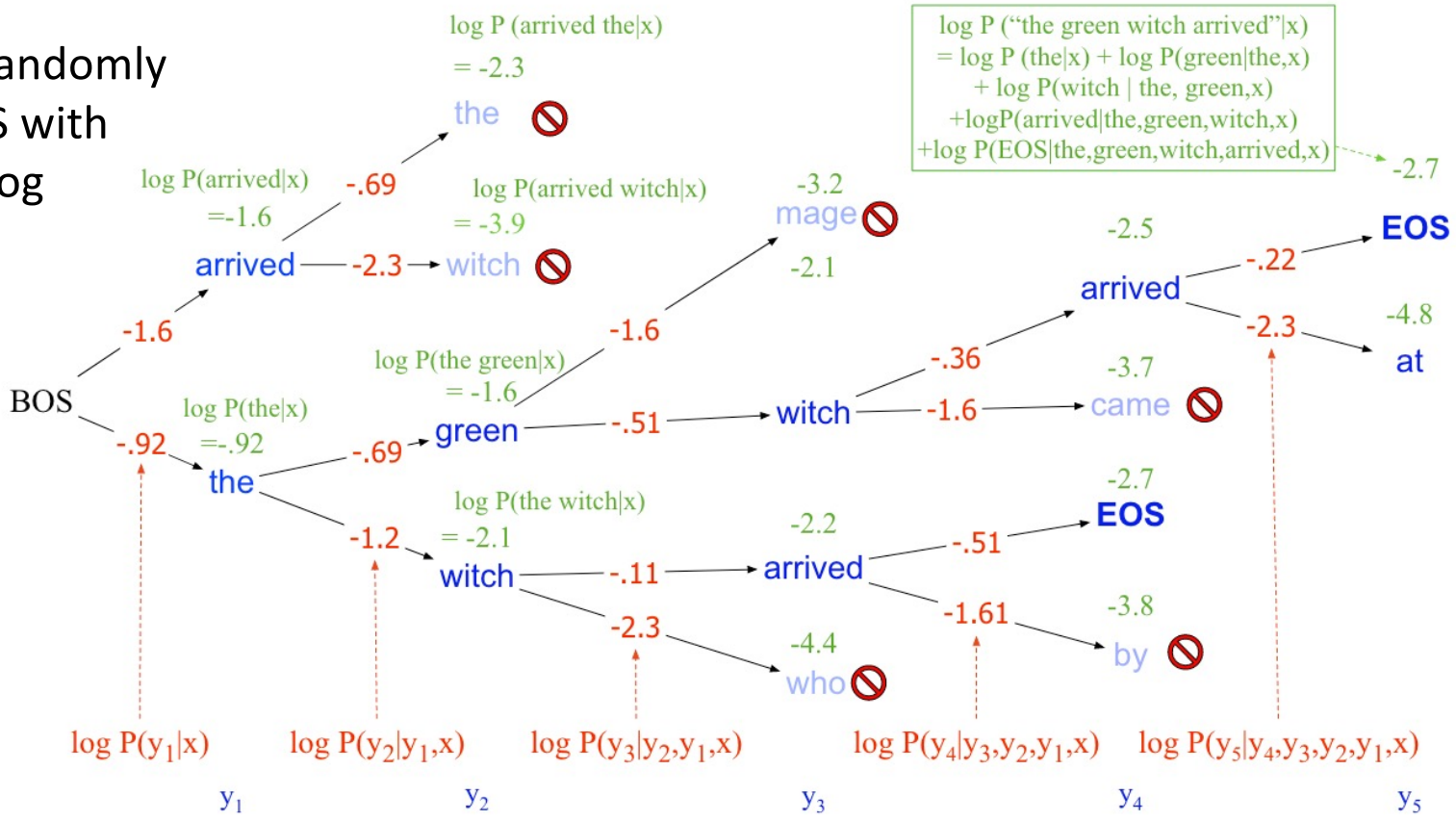


D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search

We have 2 paths randomly terminating at EOS with same cumulative log probabilities.

Randomly pick 1.



D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

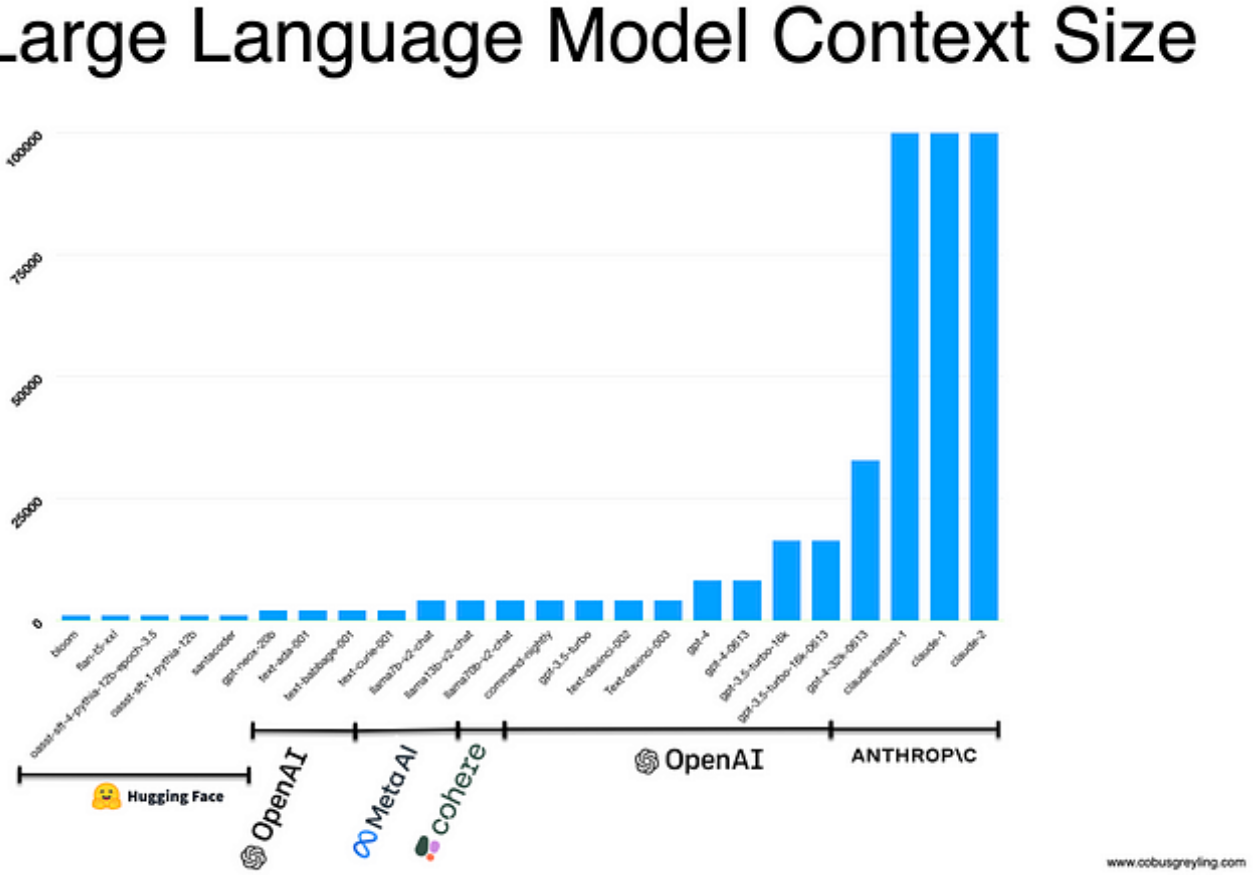
Next Token Selection

- Greedy selection
- Top-K
- Nucleus
- Beam search

Transformers for Long Sequences

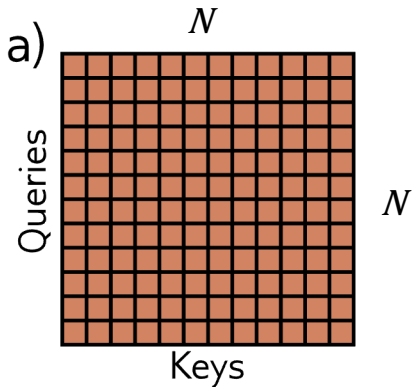
Context Length of LLMs

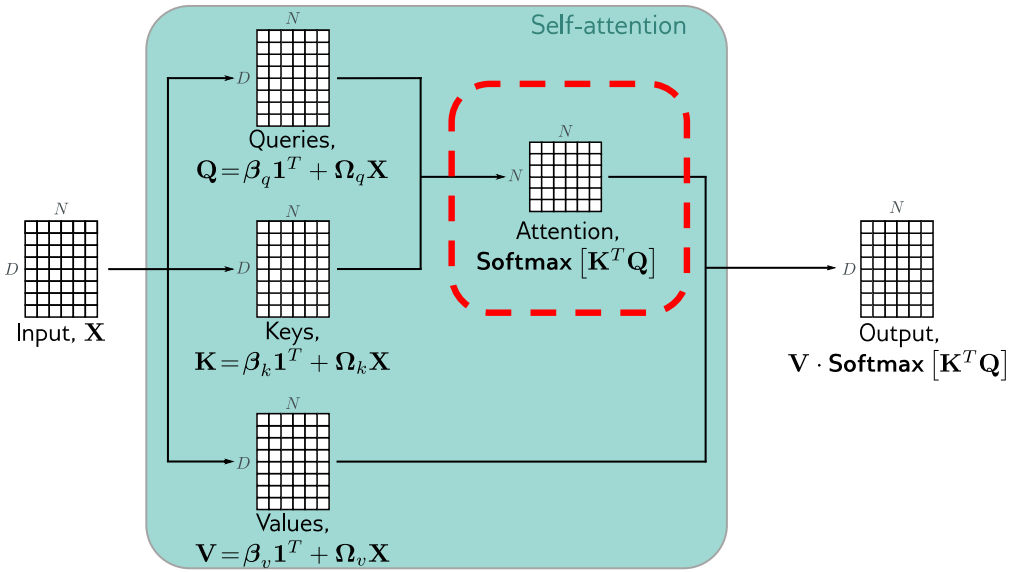
Model	Context Length
Llama 2	32K
GPT4	32K
GPT-4 Turbo, Llama 3.1	128K
Claude 3.5 Sonnet	200K
Google Gemini 1.5 Pro	Millions



https://cobusgreyling.medium.com/rag-llm-context-size-6728a2f44beb

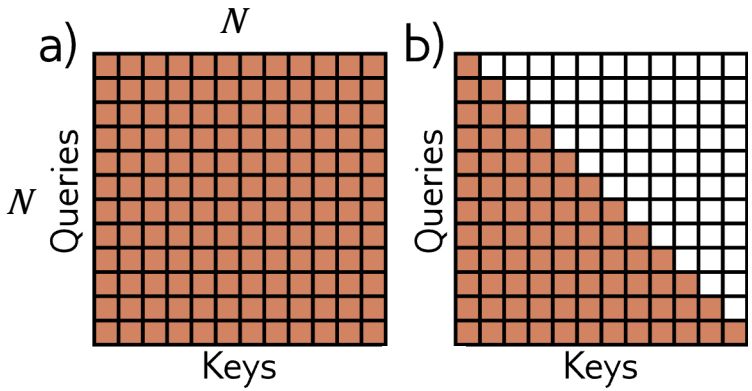
Attention Matrix

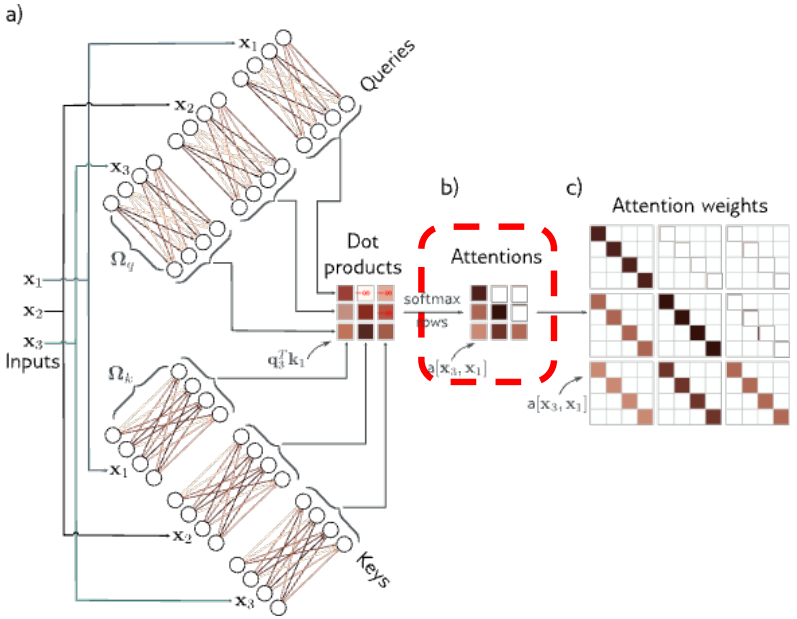




The Problem: Scales quadratically with sequence length N, e.g. N^2.

Masked Attention

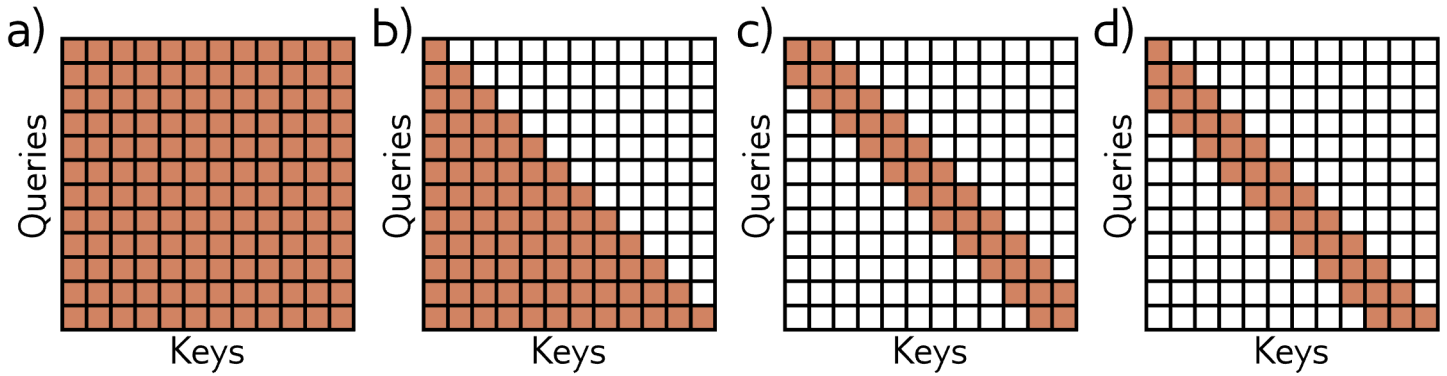




~1/2 the interactions but still scales quadratically

Use Convolutional Structure in Attention

Only attend to fixed number of neighbors



Encoder (non-causal)

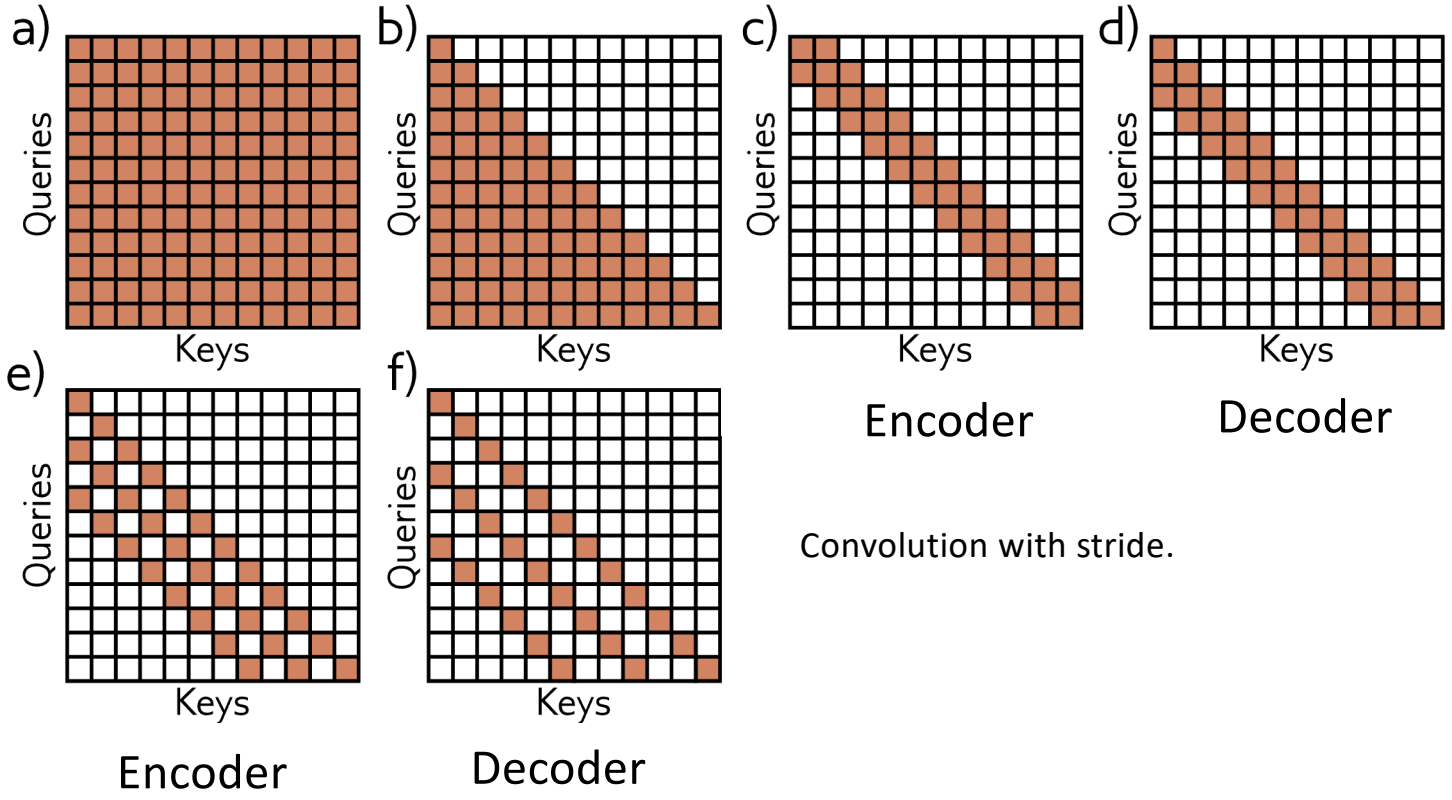
Decoder (causal)

Go from $O(N^2)$ to $O(N)$

The “receptive field” of the attention head expands across multiple transformer block layers.

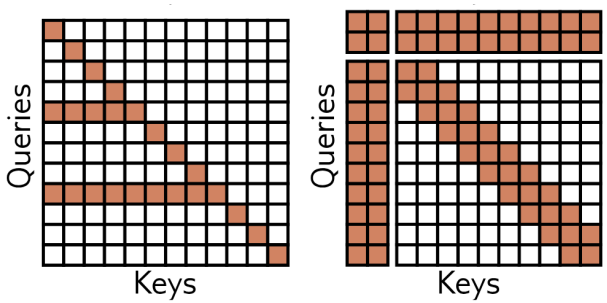
Dilated Convolutional Structures

Another way to increase the receptive field



Convolution with stride.

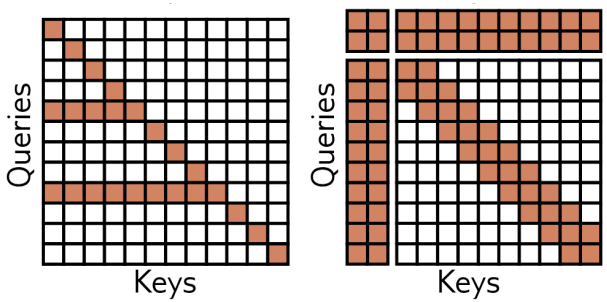
Have some tokens interact globally



Decoder Encoder

- With small neighborhood, receptive field grows slowly across layers
- Alternatively
  - Select queries interact with all previous keys (as in decoder), or
  - A few tokens that interact globally (as in encoder)
- Adding a constant number of global tokens preserves $O(N)$ complexity

Have some tokens interact globally



Used in architectures like:

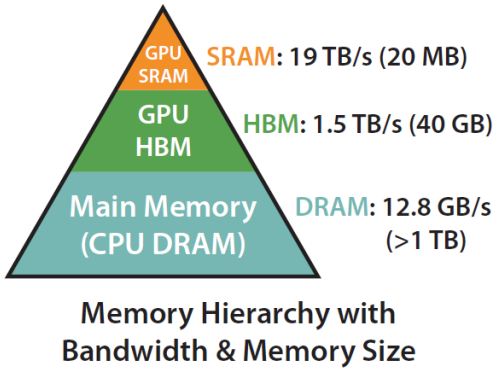
- *Sparse Transformer: Factorized Attention for Long Sequences*, Rewon Child, Scott Gray, Alec Radford, Ilya Sutskever, NeurIPS 2019, https://arxiv.org/abs/1904.10509
- *Longformer: The Long-Document Transformer*, Iz Beltagy, Matthew E. Peters, Arman Cohan, ACL 2020, https://arxiv.org/abs/2004.05150
- *BigBird: Transformers for Longer Sequences*, Manzil Zaheer et al., NeurIPS 2020, https://arxiv.org/abs/2007.14062
- *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context*, Zihang Dai et al., ACL 2019, https://arxiv.org/abs/1901.02860
- *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, Jacob Devlin et al., NAACL 2019, https://arxiv.org/abs/1810.04805

Many of Attempts at Sub-Quadratic Attention

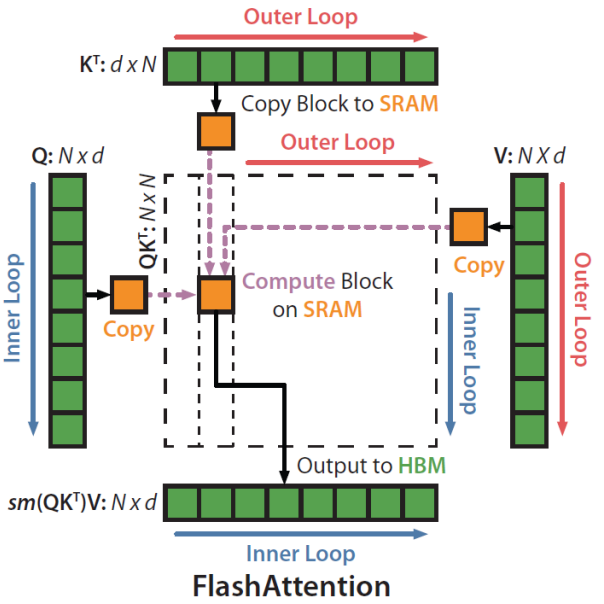
- AFAIK none in state-the-art-models
  - Many published papers claiming linear or $n \log n$ scaling with comparable performance, but none demonstrated at the same scale.
    - 10-20B parameters vs 500B parameters.

- Many practical speedups for leaner quadratic attention.
  - FlashAttention is popular.
    - Redundant computation to save on memory bandwidth (bottlenecks)

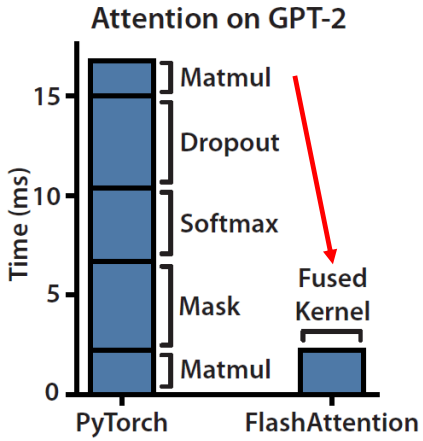
FlashAttention – IO Aware Attention Computation



Memory Hierarchy with Bandwidth & Memory Size



FlashAttention



1. In processors, near memory is faster but smaller.

2. Decompose into sub-blocks and doing more steps on each sub-block (*a.k.a. fusing kernels*)

2. 7.6x speedup on GPT2 Attention computation!!

Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." *neurips* (2022), https://arxiv.org/abs/2205.14135

Gordić, A., "ELI5: FlashAttention", Medium Post 2023, Link

Next

- Training, tuning and evaluating LLMs
- Image and multimodal transformers

Feedback

https://forms.gle/pXHM5nx1Ti9aFmpw6