# DATA 8005 Advanced Natural Language Processing

## Lecture 2: Introduction to LLMs

Fall 2024

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

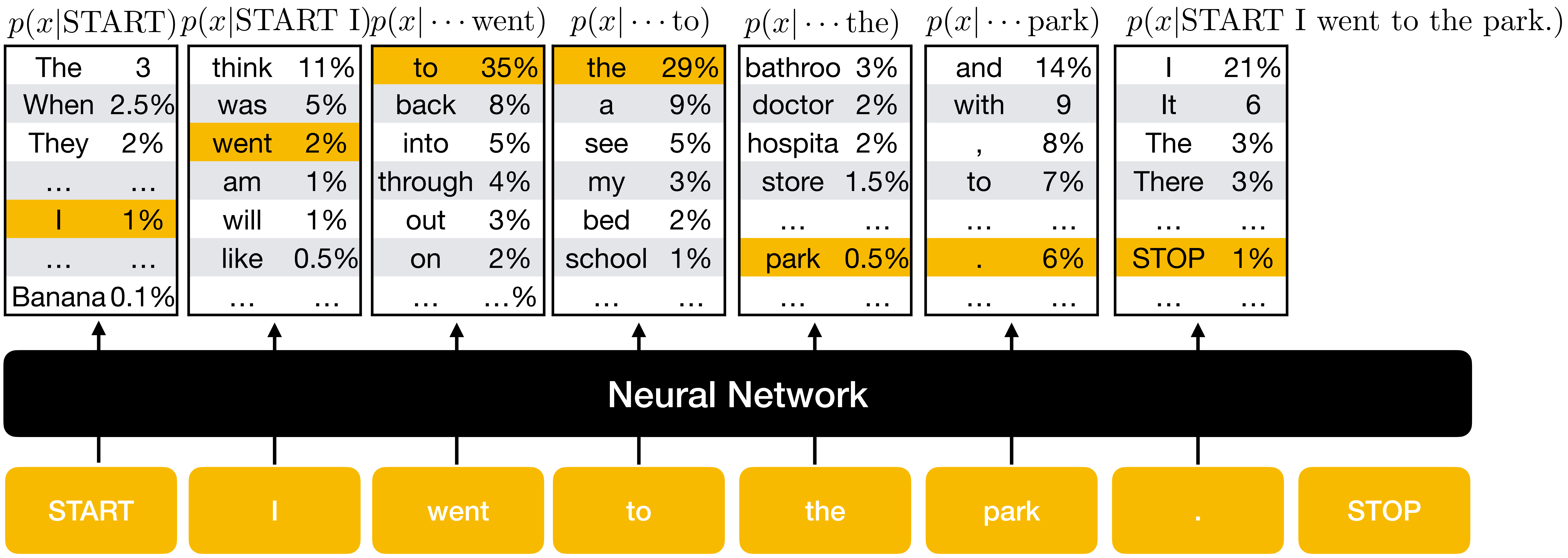
## Announcements

- Sign up for in-class presentations and final projects
  - In-class presentation: by Sep 22
  - Final projects: by Oct 4

## Neural language models: overview

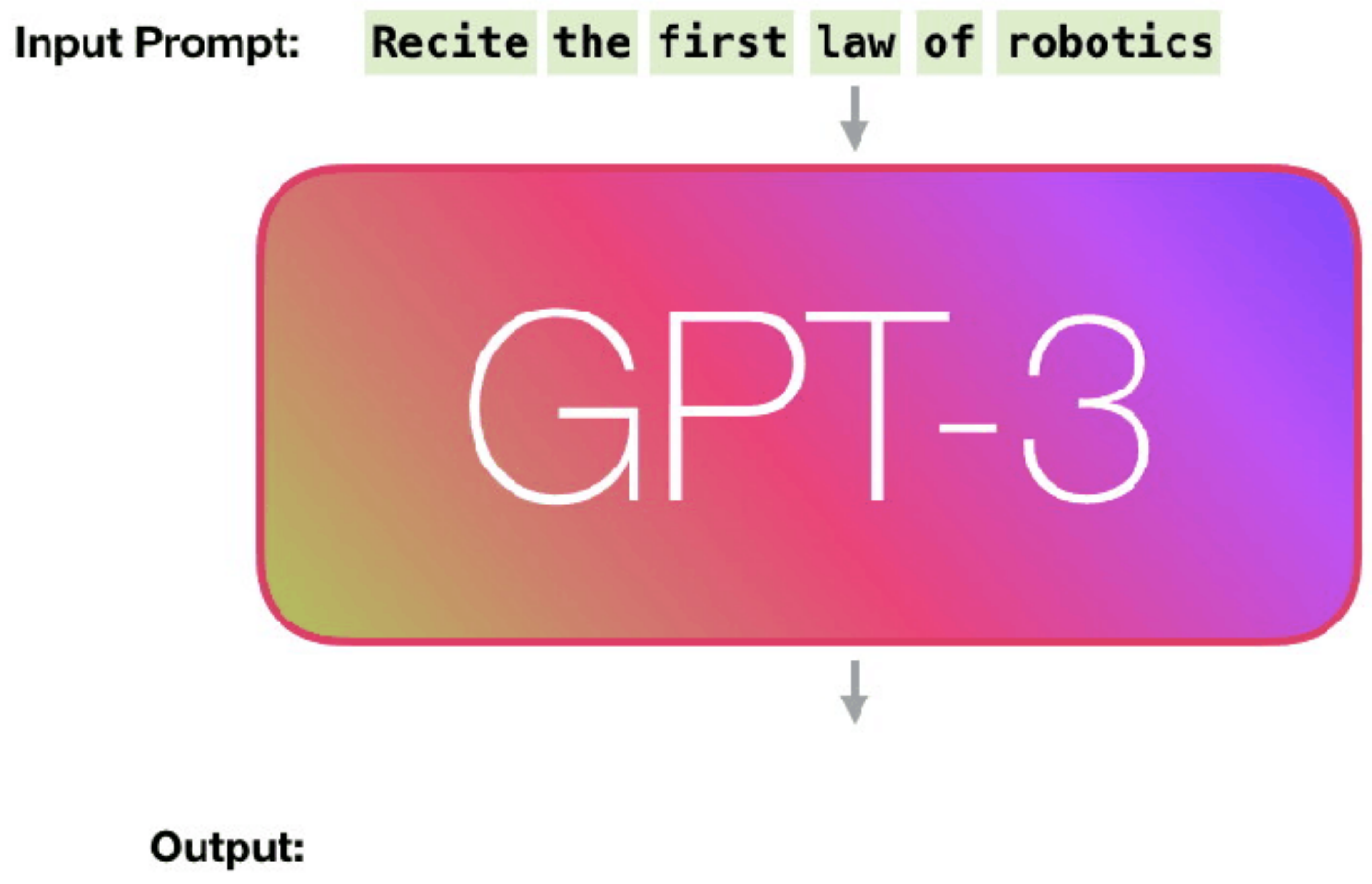
# Neural language models: inputs/outputs

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

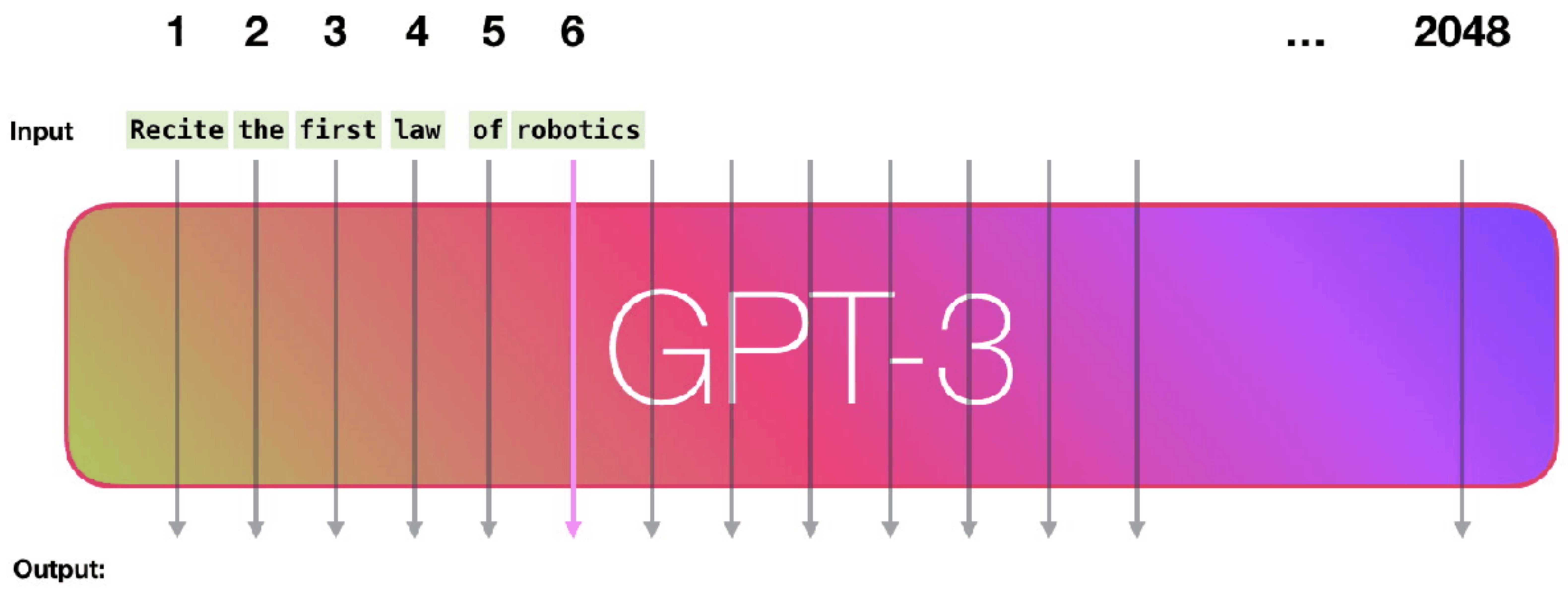


| $p(x\mid\text{START})$ | | $p(x\mid\text{START I})$ | | $p(x\mid\cdots\text{went})$ | | $p(x\mid\cdots\text{to})$ | | $p(x\mid\cdots\text{the})$ | | $p(x\mid\cdots\text{park})$ | | $p(x\mid\text{START I went to the park.})$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| The | 3 | think | 11% | to | 35% | the | 29% | bathroo | 3% | and | 14% | I | 21% |
| When | 2.5% | was | 5% | back | 8% | a | 9% | doctor | 2% | with | 9 | It | 6 |
| They | 2% | went | 2% | into | 5% | see | 5% | hospita | 2% | , | 8% | The | 3% |
| … | … | am | 1% | through | 4% | my | 3% | store | 1.5% | to | 7% | There | 3% |
| I | 1% | will | 1% | out | 3% | bed | 2% | … | … | … | … | … | … |
| … | … | like | 0.5% | on | 2% | school | 1% | park | 0.5% | . | 6% | STOP | 1% |
| Banana | 0.1% | … | … | … | …% | … | … | … | … | … | … | … | … |

# Usage example: GPT-3



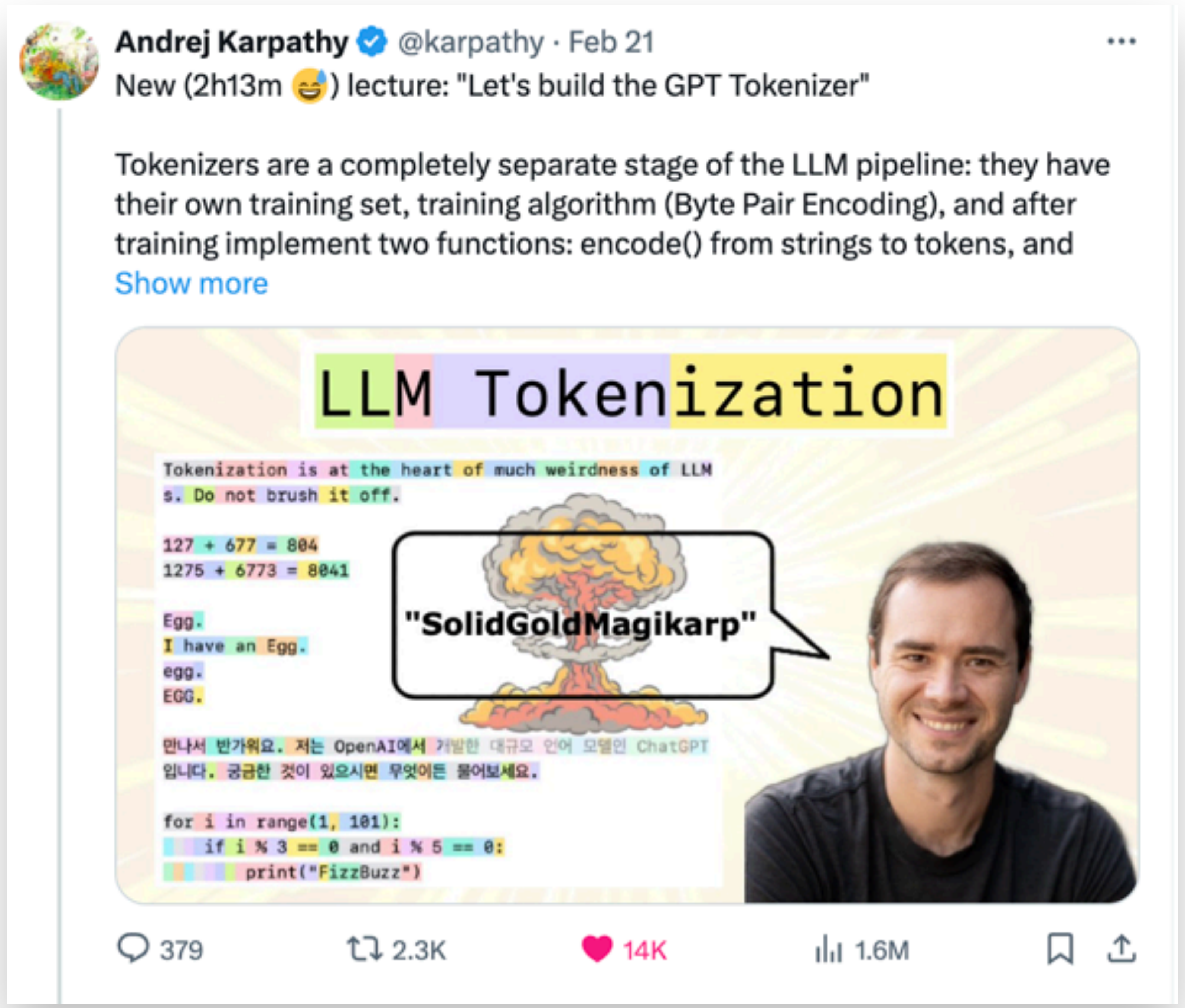
# Usage example: GPT-3



https://jalammar.github.io/how-gpt3-works-visualizations-animations/

## Neural language models: tokenization

# Why is tokenization important?



**Andrej Karpathy** @karpathy · Feb 21

New (2h13m 😅) lecture: "Let's build the GPT Tokenizer"

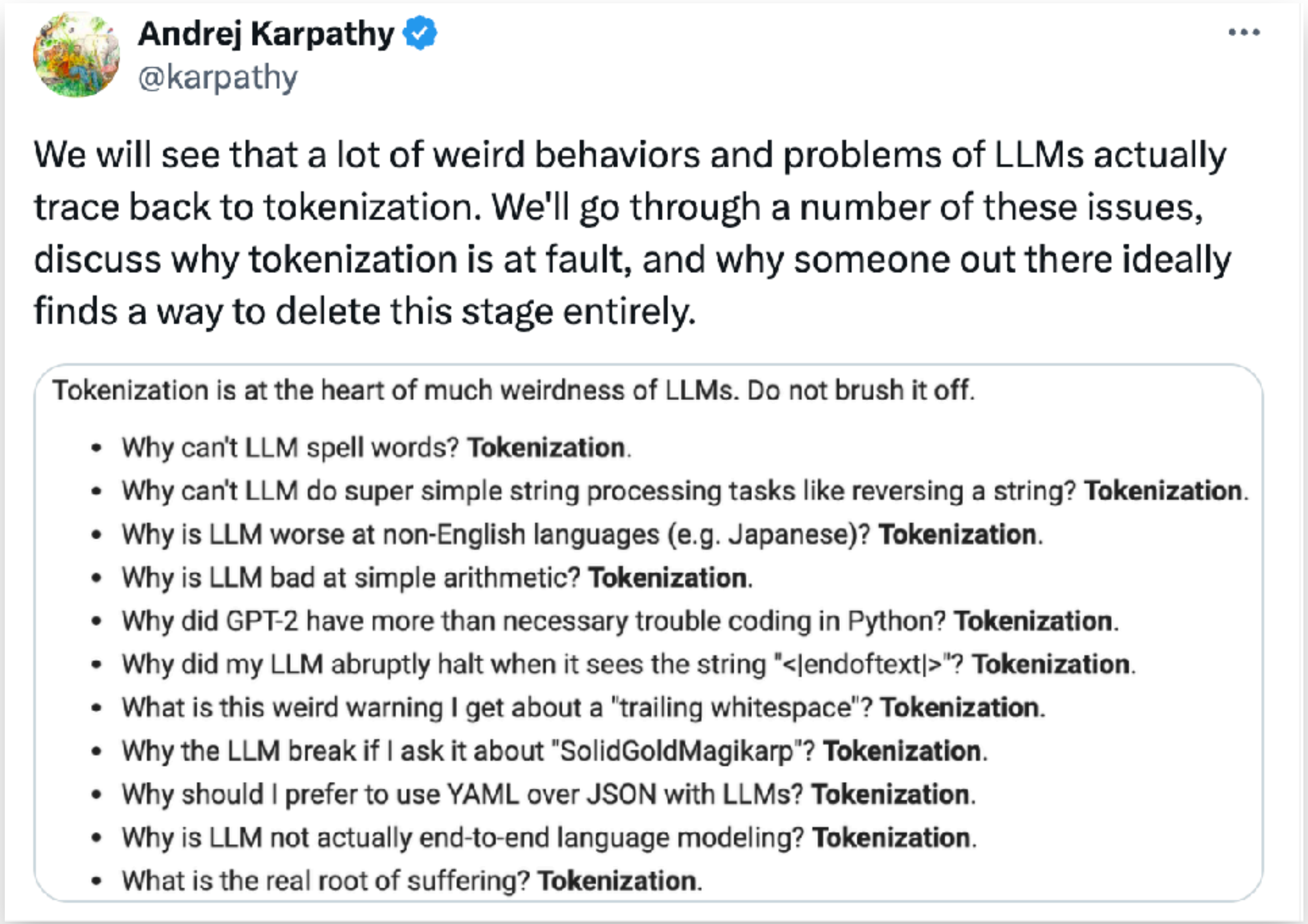
Tokenizers are a completely separate stage of the LLM pipeline: they have their own training set, training algorithm (Byte Pair Encoding), and after training implement two functions: encode() from strings to tokens, and

Show more

LLM Tokenization

```
Tokenization is at the heart of much weirdness of LLM
s. Do not brush it off.

127 + 677 = 804
1275 + 6773 = 8041

Egg.
I have an Egg.
egg.
EGG.

만나서 반가워요. 저는 OpenAI에서 개발한 대규모 언어 모델인 ChatGPT
입니다. 궁금한 것이 있으시면 무엇이든 물어보세요.

for i in range(1, 101):
    if i % 3 == 0 and i % 5 == 0:
        print("FizzBuzz")
```

379 · 2.3K · 14K · 1.6M



**Andrej Karpathy** @karpathy

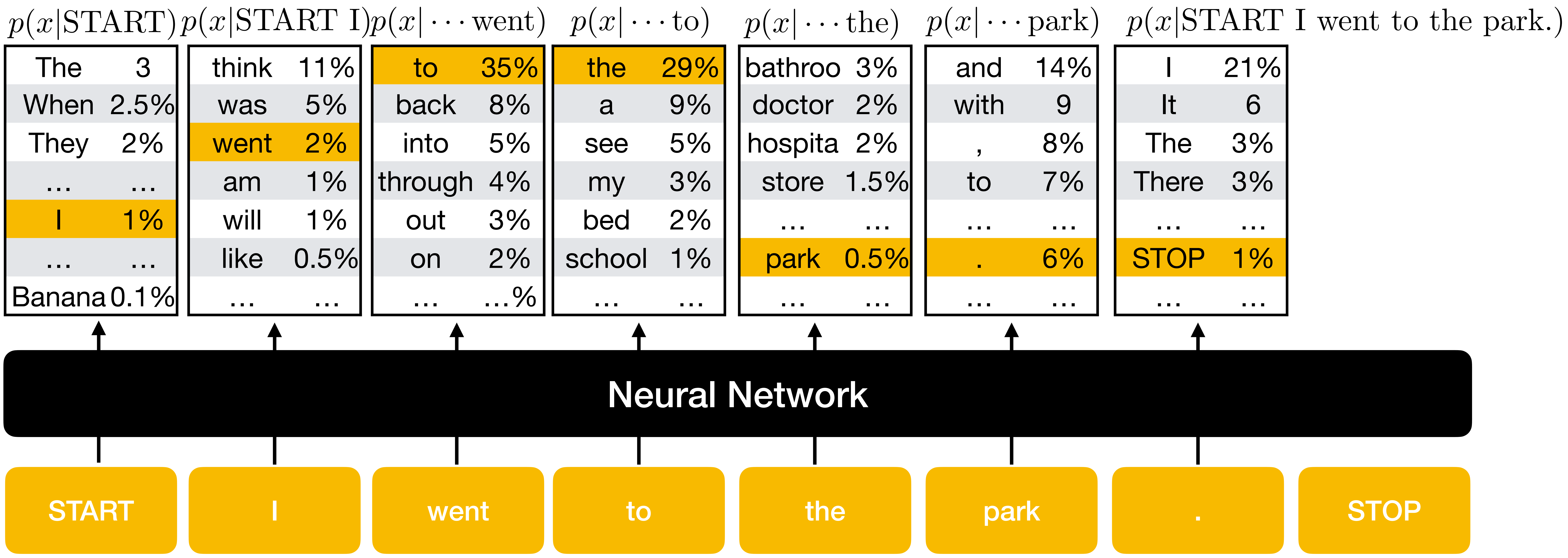
We will see that a lot of weird behaviors and problems of LLMs actually trace back to tokenization. We'll go through a number of these issues, discuss why tokenization is at fault, and why someone out there ideally finds a way to delete this stage entirely.

Tokenization is at the heart of much weirdness of LLMs. Do not brush it off.

- 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 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**.

# Neural language models: inputs/outputs

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



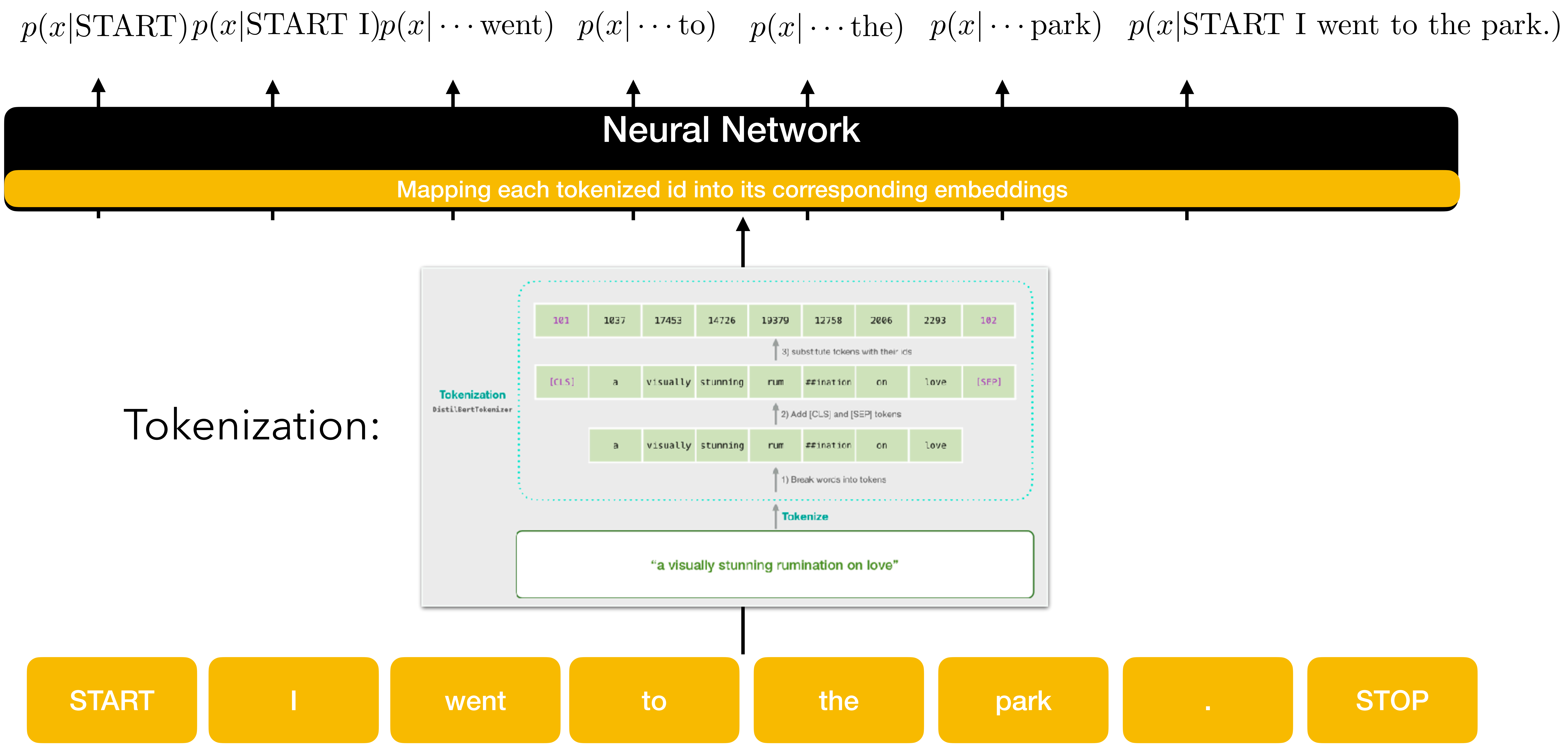
| $p(x\|\text{START})$ | | $p(x\|\text{START I})$ | | $p(x\|\cdots\text{went})$ | | $p(x\|\cdots\text{to})$ | | $p(x\|\cdots\text{the})$ | | $p(x\|\cdots\text{park})$ | | $p(x\|\text{START I went to the park.})$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| The | 3 | think | 11% | to | 35% | the | 29% | bathroo | 3% | and | 14% | I | 21% |
| When | 2.5% | was | 5% | back | 8% | a | 9% | doctor | 2% | with | 9 | It | 6 |
| They | 2% | went | 2% | into | 5% | see | 5% | hospita | 2% | , | 8% | The | 3% |
| … | … | am | 1% | through | 4% | my | 3% | store | 1.5% | to | 7% | There | 3% |
| I | 1% | will | 1% | out | 3% | bed | 2% | … | … | … | … | … | … |
| … | … | like | 0.5% | on | 2% | school | 1% | park | 0.5% | . | 6% | STOP | 1% |
| Banana | 0.1% | … | … | … | …% | … | … | … | … | … | … | … | … |

## Neural language models: input vectors

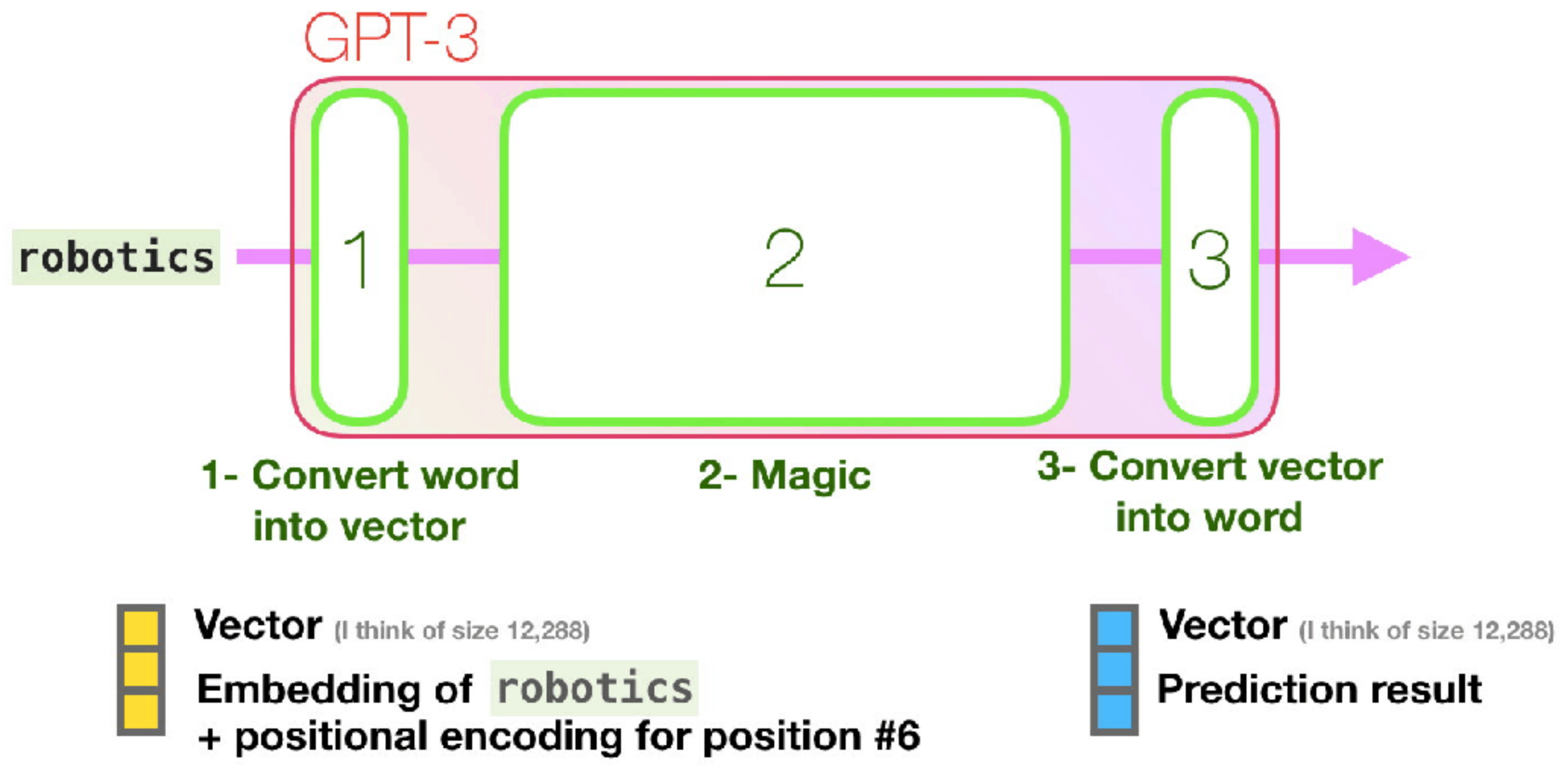
*But neural networks take in real-valued vectors, not words…*

- Use one-hot or learned embeddings to map from words to vectors!
  - Learned embeddings become part of parameters $\theta$

# Tokenization to input vectors

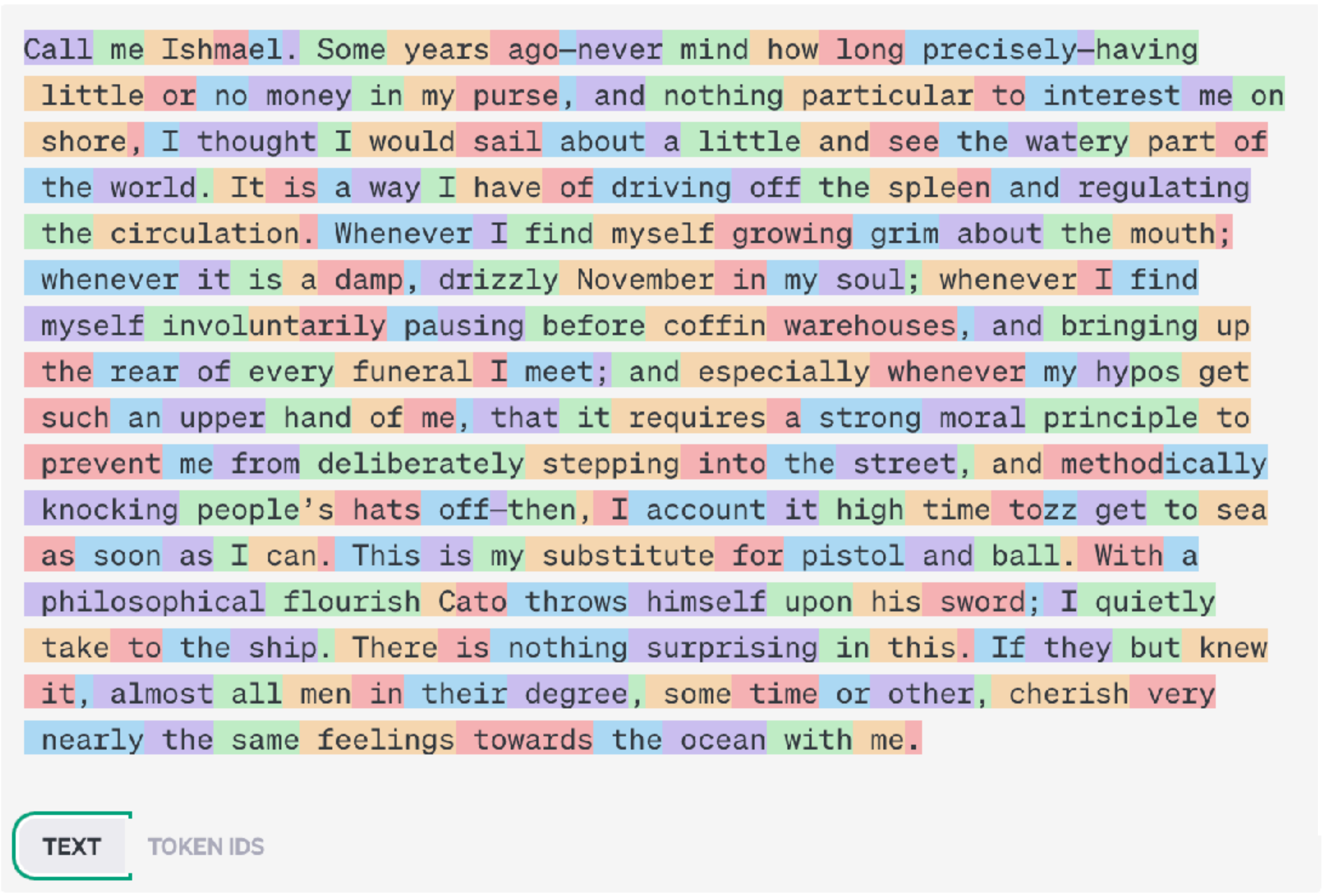

$p(x|\text{START})$
$p(x|\text{START I})$
$p(x|\cdots\text{went})$
$p(x|\cdots\text{to})$
$p(x|\cdots\text{the})$
$p(x|\cdots\text{park})$
$p(x|\text{START I went to the park.})$
Neural Network
Mapping each tokenized id into its corresponding embeddings
Tokenization:
Tokenization
DistilBertTokenizer
101 1037 17453 14726 19379 12758 2006 2293 102
3) substitute tokens with their ids
[CLS] a visually stunning rum ##ination on love [SEP]
2) Add [CLS] and [SEP] tokens
a visually stunning rum ##ination on love
1) Break words into tokens
Tokenize
“a visually stunning rumination on love”
START
I
went
to
the
park
.
STOP

# Usage example: GPT-3

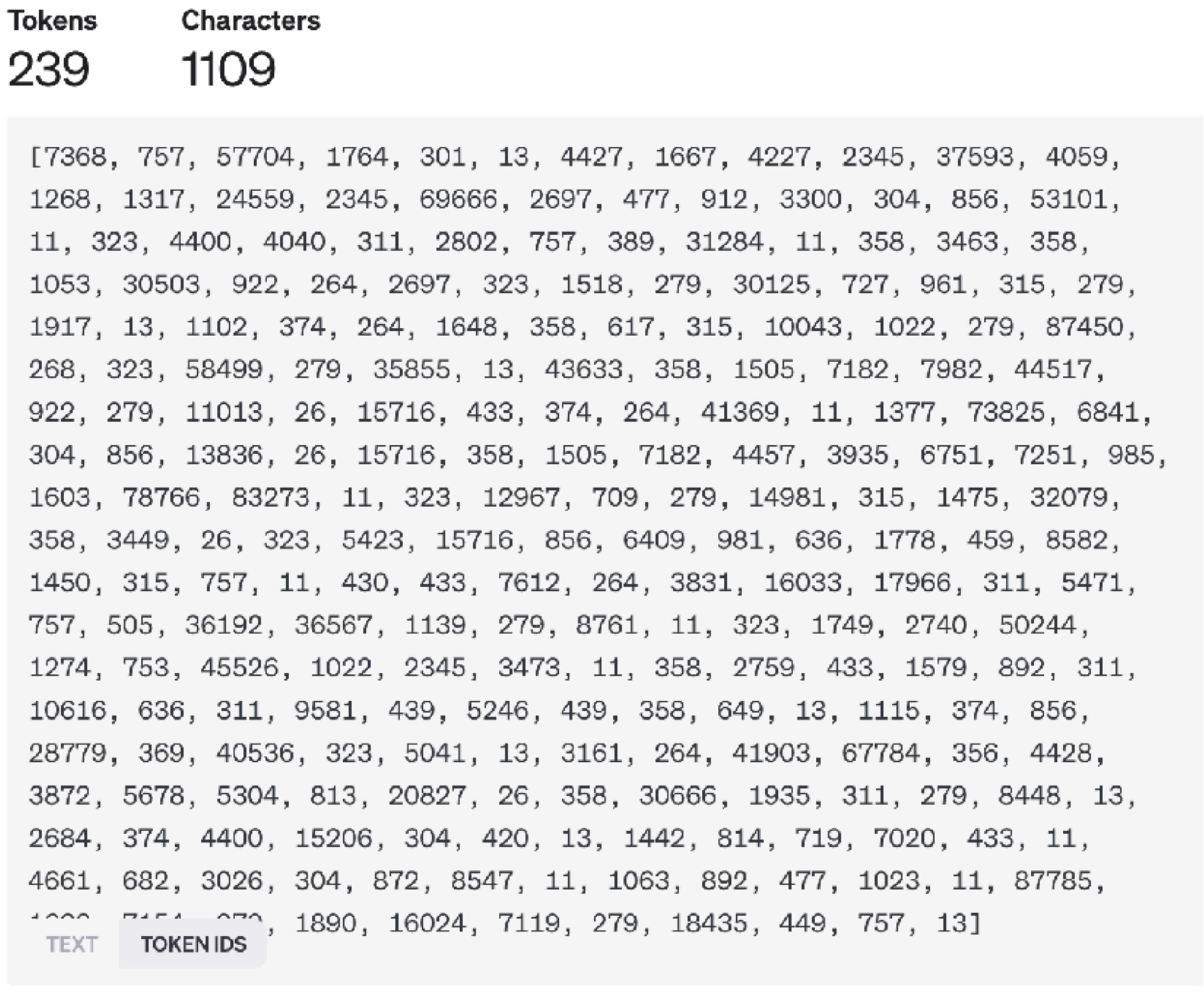

GPT-3
robotics
1
2
3
1- Convert word into vector
2- Magic
3- Convert vector into word
Vector (I think of size 12,288)
Embedding of robotics + positional encoding for position #6
Vector (I think of size 12,288)
Prediction result

https://jalammar.github.io/how-gpt3-works-visualizations-animations/

# 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.
```



| Tokens | Characters |
|---|---|
| 239 | 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]
```

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~~ Subword tokenization!

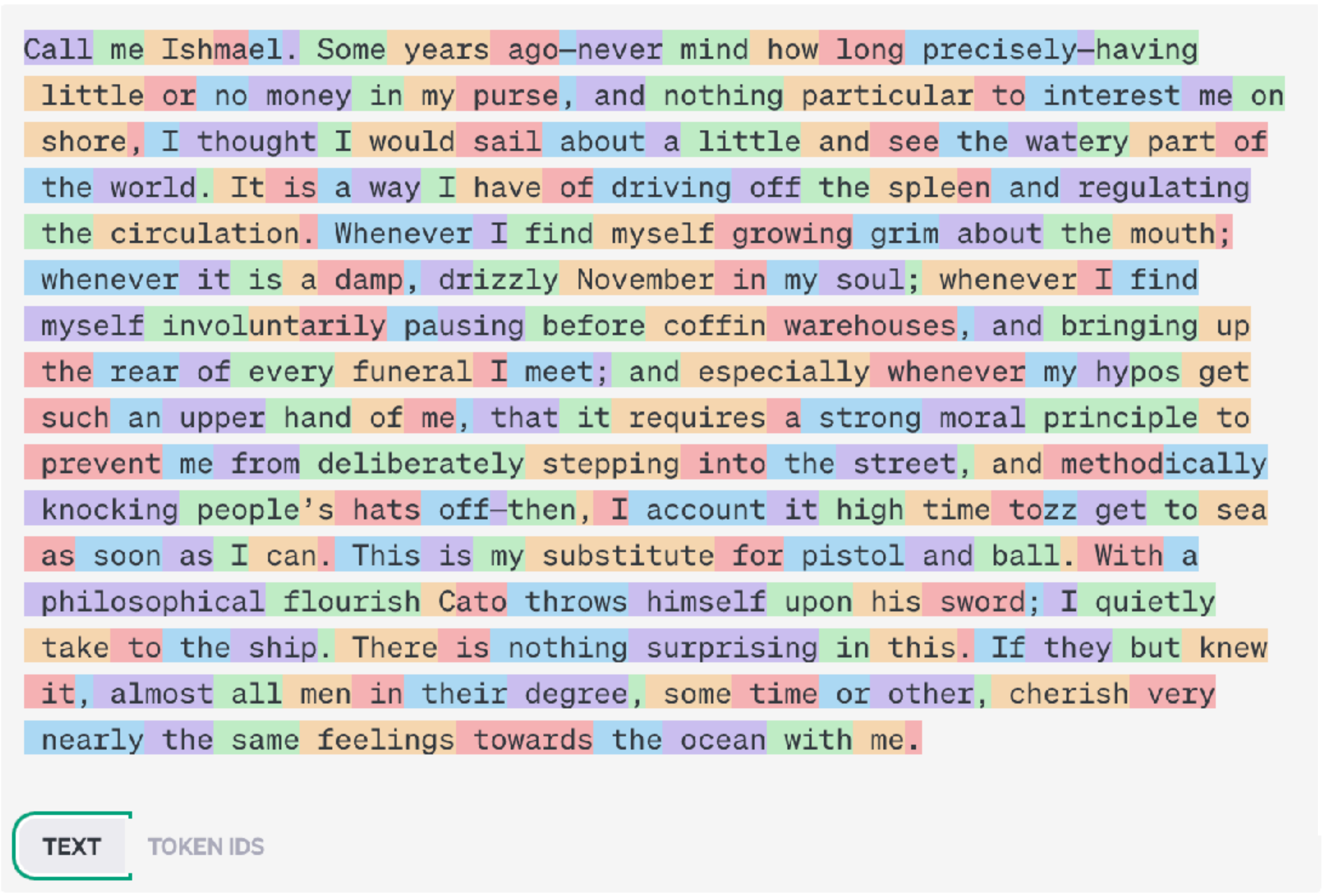
How can we combine the high coverage of character-level representation with the efficiency of word-level representation?

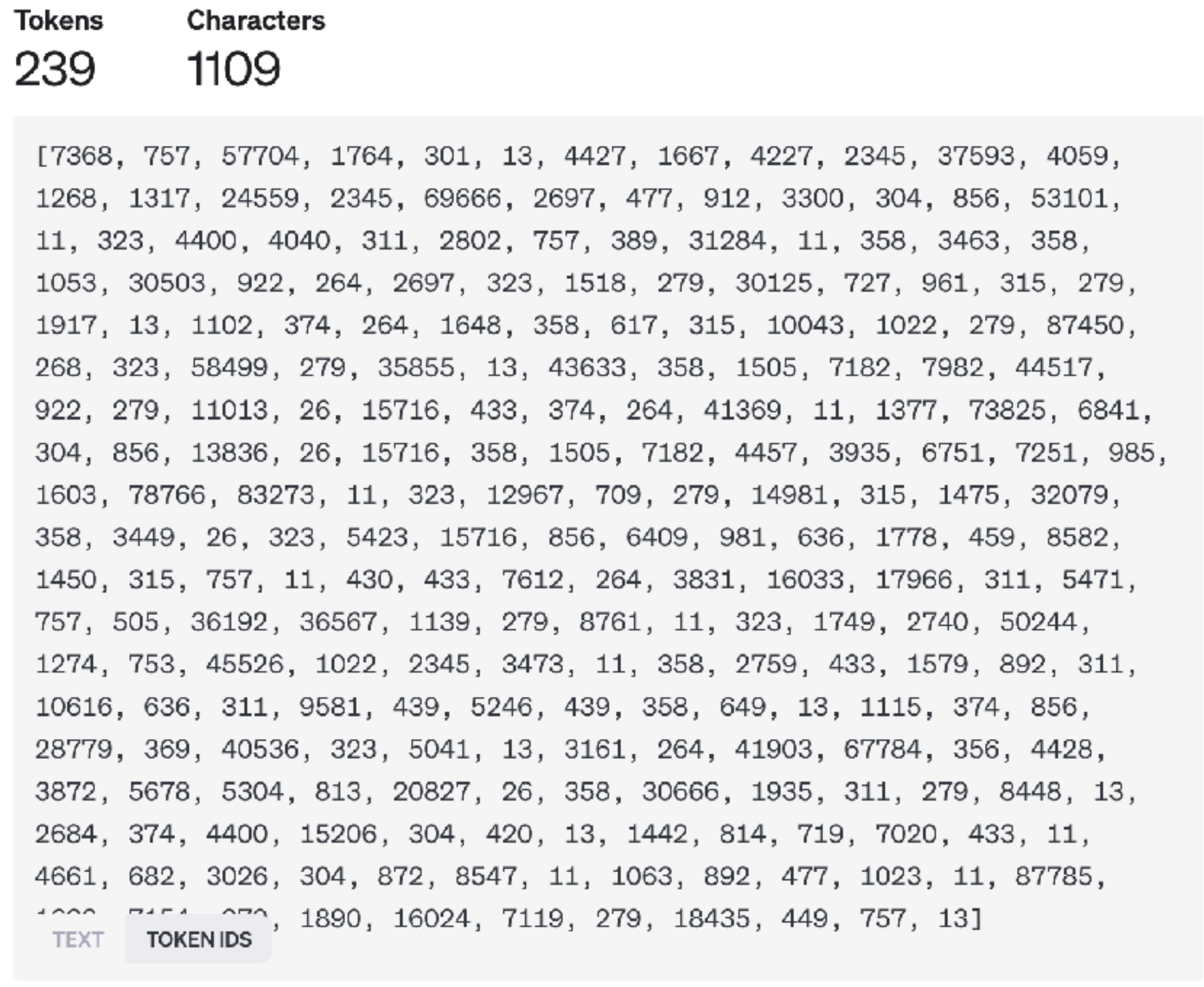
**Subword tokenization! (e.g., Byte-Pair Encoding)**

- 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.


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]
```

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

**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 $\mathcal{V}$

$$\mathcal{D} = \{\text{“i”}, \text{“ hug”}, \text{“ pugs”}, \text{“hugging”}, \text{“ pugs”}, \text{“ is”}, \text{“ fun”}, \text{“i”}, \text{“ make”}, \text{“ puns”}\}$$

$$\mathcal{V} = \{\text{‘ ’}, \text{‘a’}, \text{‘e’}, \text{‘f’}, \text{‘g’}, \text{‘h’}, \text{‘i’}, \text{‘k’}, \text{‘m’}, \text{‘n’}, \text{‘p’}, \text{‘s’}, \text{‘u’}\}, |\mathcal{V}| = 13$$

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

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 $\mathcal{V}$

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

| 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* $|\mathcal{V}| = N \ldots$

$$\begin{aligned}\mathcal{D} = \{\, &[\text{‘i’}], [\text{‘ hug’}], [\text{‘ pugs’}], \\ &[\text{‘hug’}, \text{‘g’}, \text{‘i’}, \text{‘n’}, \text{‘g’}], [\text{‘ pugs’}], \\ &[\text{‘ ’}, \text{‘i’}, \text{‘s’}], [\text{‘ ’}, \text{‘f’}, \text{‘un’}], [\text{‘i’}], \\ &[\text{‘ ’}, \text{‘m’}, \text{‘a’}, \text{‘k’}, \text{‘e’}], [\text{‘ p’}, \text{‘un’}, \text{‘s’}]\}\end{aligned}$$

$$\begin{aligned}\mathcal{V} = \{&\text{‘ ’}, \text{‘a’}, \text{‘e’}, \text{‘f’}, \text{‘g’}, \text{‘h’}, \text{‘i’}, \text{‘k’}, \text{‘m’}, \text{‘n’}, \text{‘p’}, \text{‘s’}, \text{‘u’}, \\ &\text{‘ug’}, \text{‘ p’}, \text{‘hug’}, \text{‘ pug’}, \text{‘ pugs’}, \text{‘un’}, \text{‘ hug’}\},\end{aligned}$$

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

CHANGES FROM START

# Byte-pair encoding: example

CHANGES FROM START

**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?

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

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

*(as tokens indices)*

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

# 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]

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

## Byte-pair encoding: tokenization/encoding

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

- 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

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

**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

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

**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 $s$

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

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

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

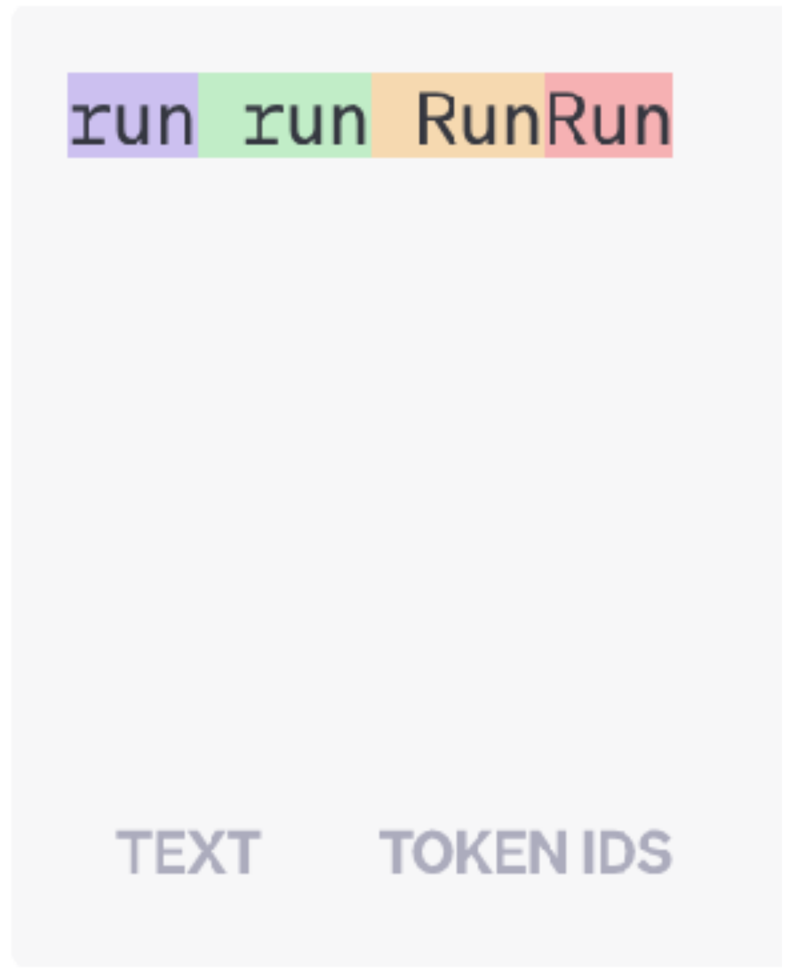
$$\text{Decode}([9, 7, 12, 12, 6, 2, 11, 3, 10, 10, 3, 12, 12]) = \text{``misshapenness''}$$

## Byte-pair encoding: properties

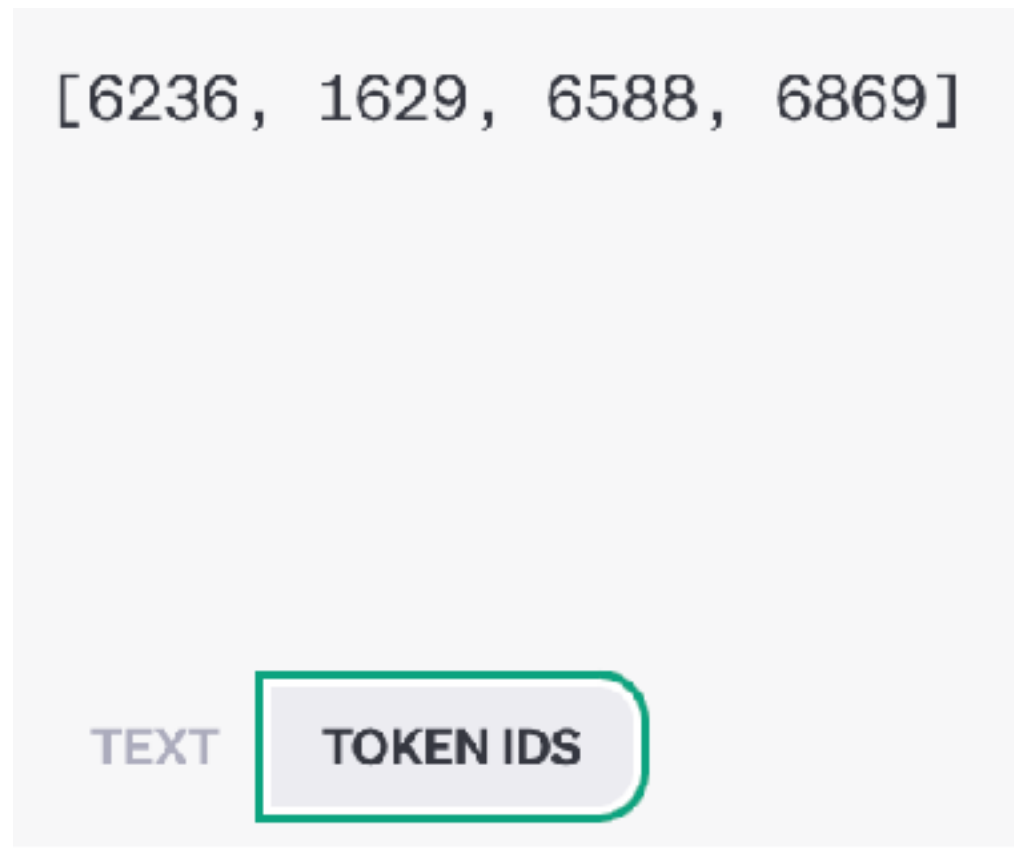
- Efficient to run (greedy vs. global optimization)
- Lossless compression
- Potentially some shared representations - e.g., the token “hug” could be used both in “hug” and “hugging”

# 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”



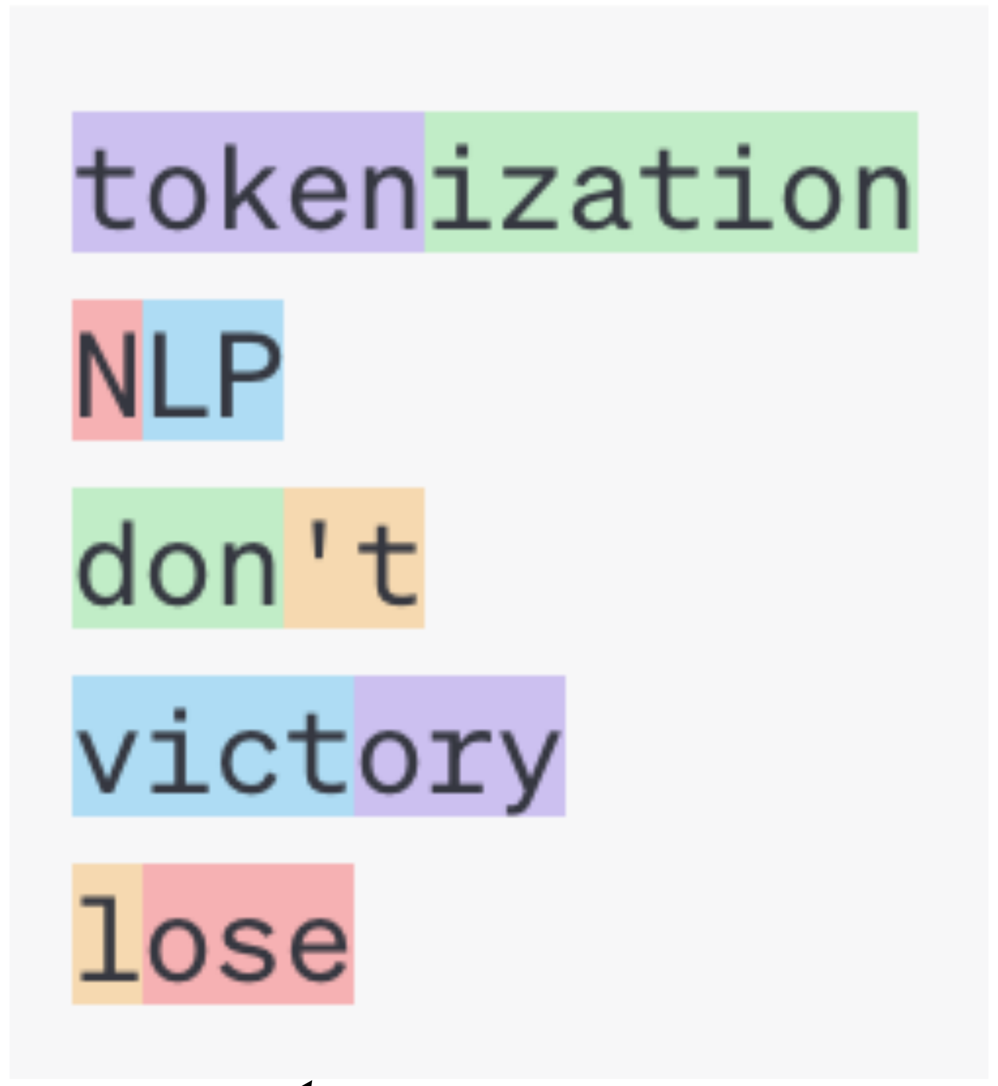
```
run run RunRun
```



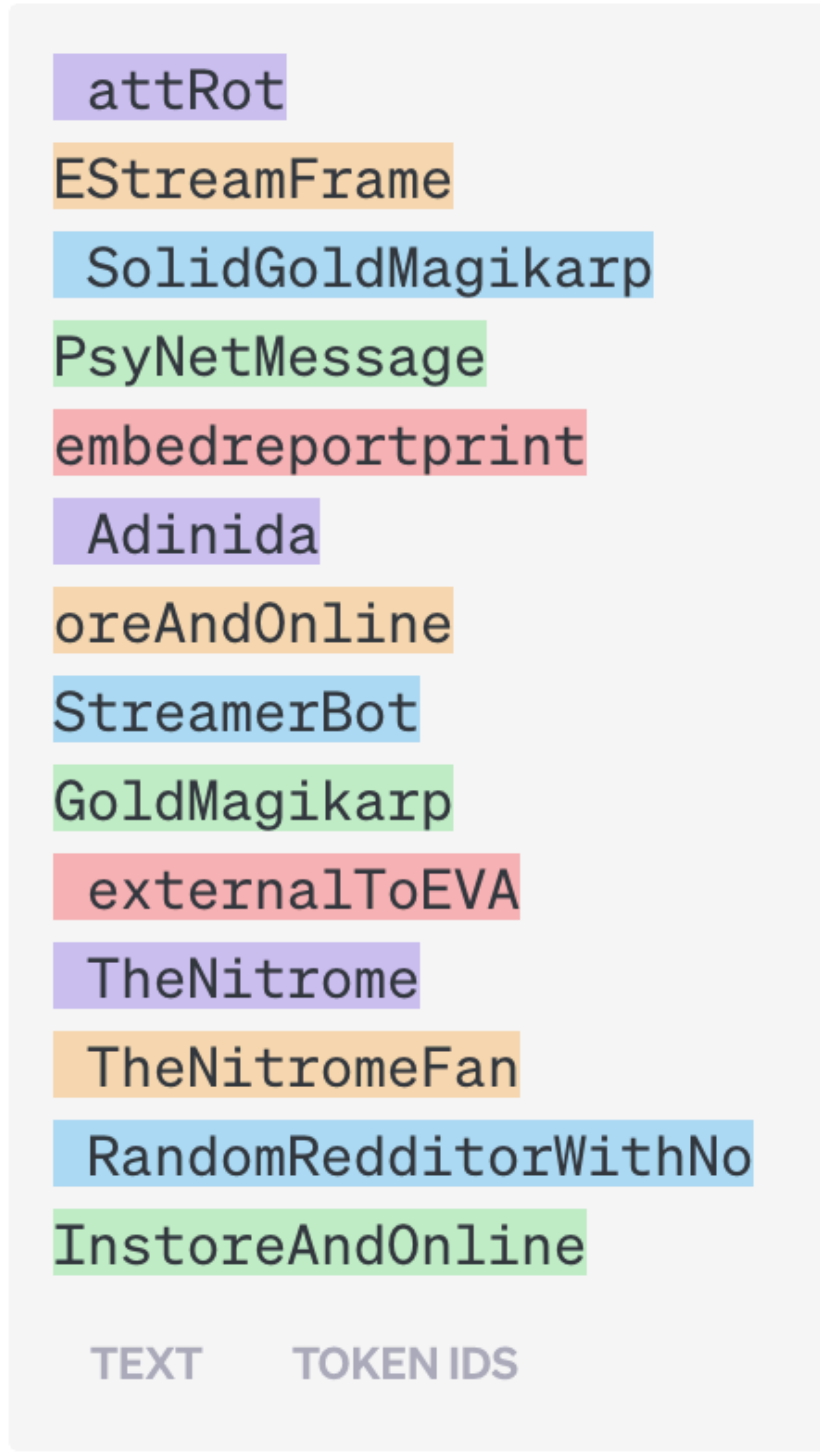
```
[6236, 1629, 6588, 6869]
```

# 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!



```
tokenization
NLP
don't
victory
lose
```



```
attRot
EStreamFrame
SolidGoldMagikarp
PsyNetMessage
embedreportprint
Adinida
oreAndOnline
StreamerBot
GoldMagikarp
externalToEVA
TheNitrome
TheNitromeFan
RandomRedditorWithNo
InstoreAndOnline
```

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

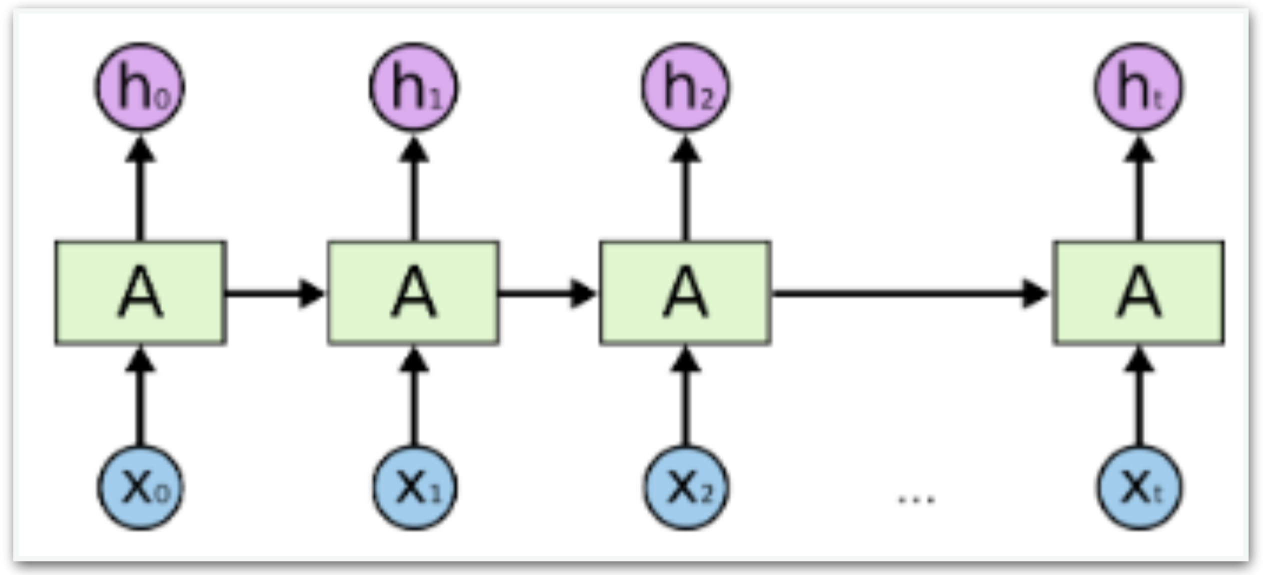
# Other tokenization variants

| Model/Tokenizer | Objective | Spaces part of token? | Pre-tokenization | Smallest unit |
|---|---|---|---|---|
| GPT | BPE | No | Yes | Character-level |
| GPT-2/3/4, ChatGPT, Llama(2), Falcon, … | BPE | Yes | Yes | Byte-level |
| Jurassic | BPE | Yes | No. “SentencePiece” - treat whitespace like char | Byte-level |
| Bert, DistilBert, Electra | WordPiece | No | Yes | Character-level |
| T5, ALBERT, XLNet, Marian | Unigram | Yes | No. “SentencePiece” - treat whitespace like char* | Character-level |

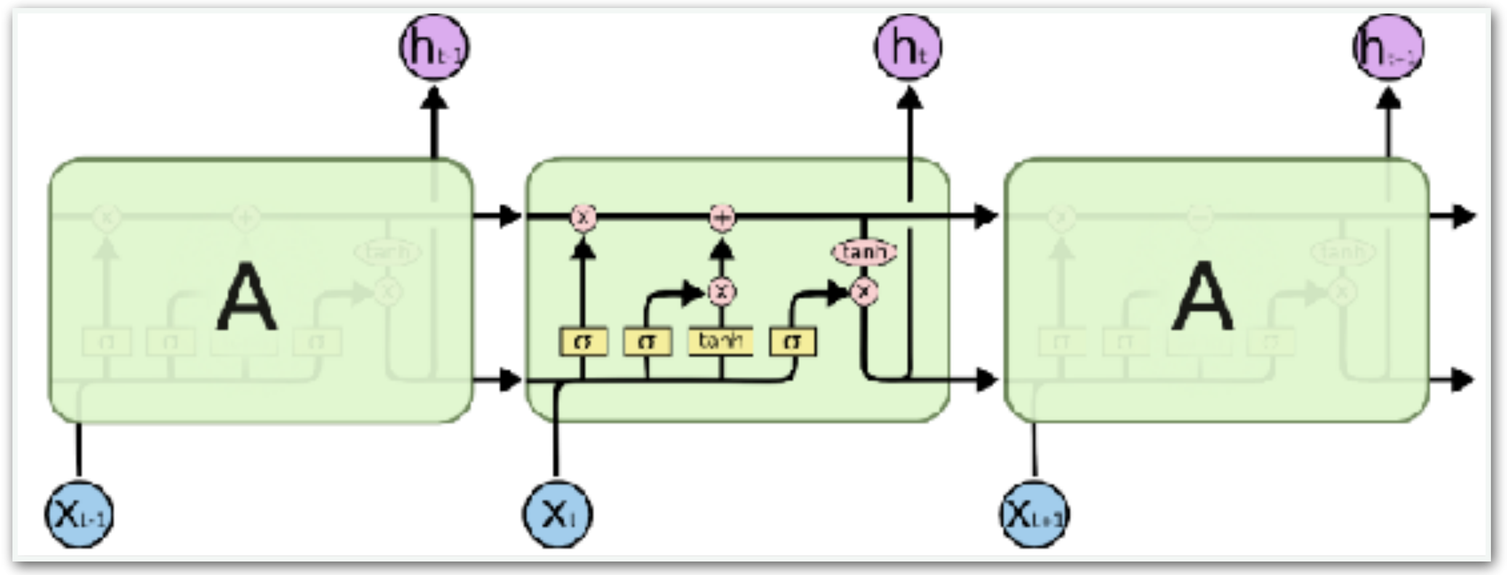
*For non-English languages

## Neural language models: transformers

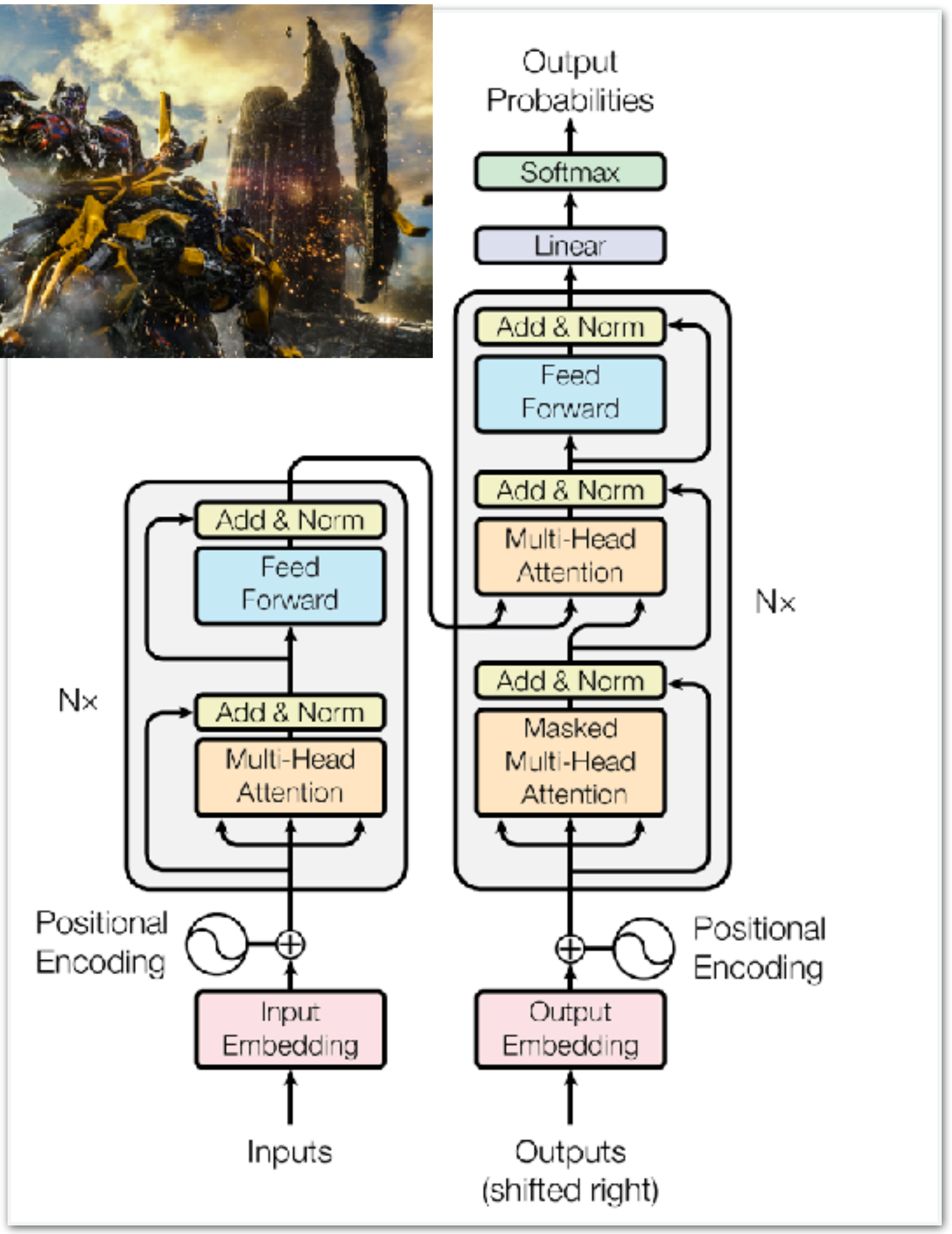
# Neural language models: neural networks



RNNs

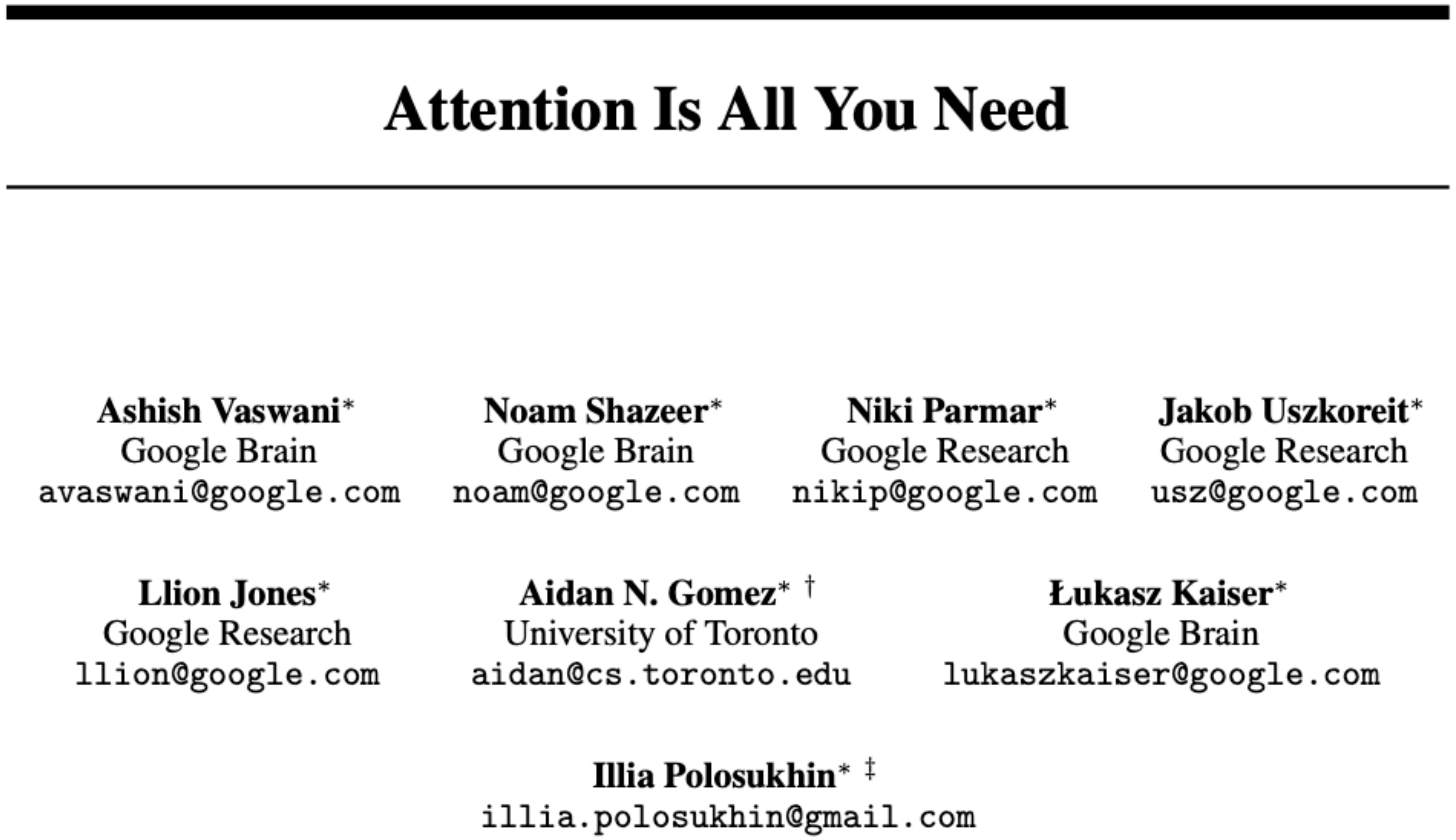


LSTM



Transformers

## Transformers



**Attention Is All You Need**

**Ashish Vaswani**[*]
Google Brain
`avaswani@google.com`

**Noam Shazeer**[*]
Google Brain
`noam@google.com`

**Niki Parmar**[*]
Google Research
`nikip@google.com`

**Jakob Uszkoreit**[*]
Google Research
`usz@google.com`

**Llion Jones**[*]
Google Research
`llion@google.com`

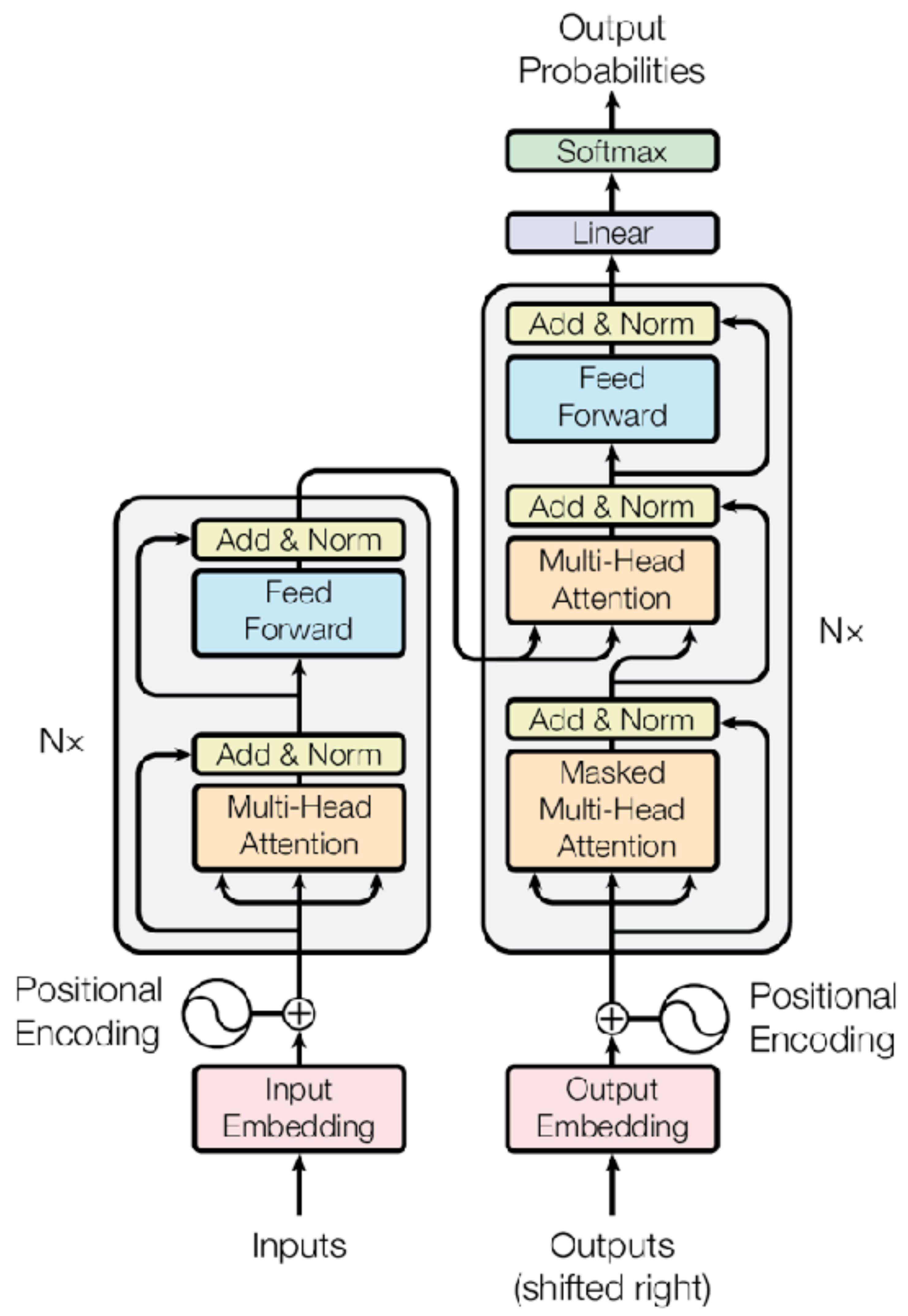
**Aidan N. Gomez**[*] [†]
University of Toronto
`aidan@cs.toronto.edu`

**Łukasz Kaiser**[*]
Google Brain
`lukaszkaiser@google.com`

**Illia Polosukhin**[*] [‡]
`illia.polosukhin@gmail.com`

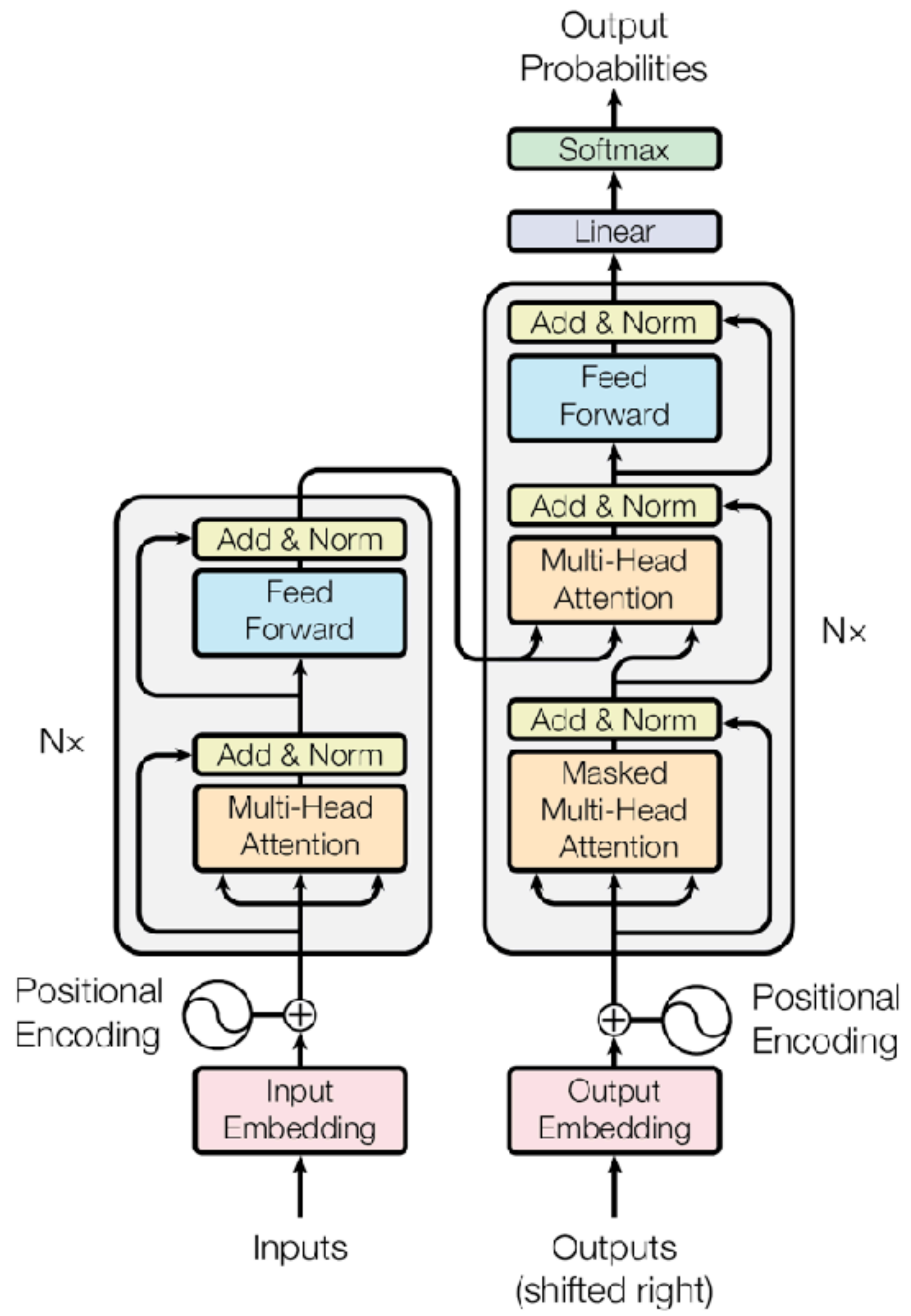
(Vaswani et al., 2017)

# Transformer encoder-decoder



- Transformer encoder + Transformer decoder
- First designed and experimented on NMT

# Transformer encoder-decoder



- Transformer encoder = a stack of **encoder layers**
- Transformer decoder = a stack of **decoder layers**

**Transformer encoder**: BERT, RoBERTa, ELECTRA

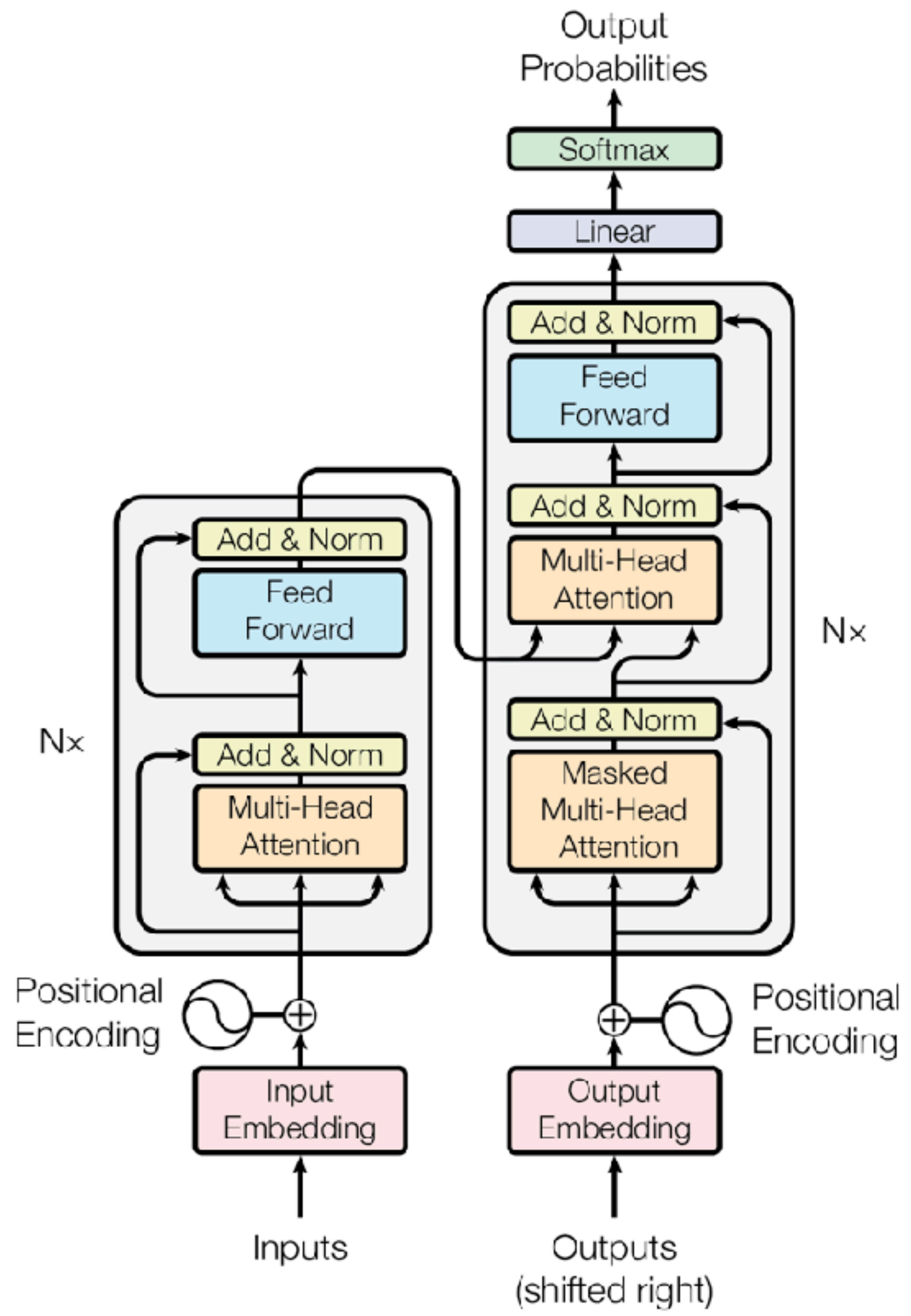
**Transformer decoder**: GPT-3, ChatGPT, Palm

**Transformer encoder-decoder**: T5, BART

- Key innovation: **multi-head, self-attention**
- Transformers don't have any recurrence structures!

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \in \mathbb{R}^h$$

# Transformers: roadmap



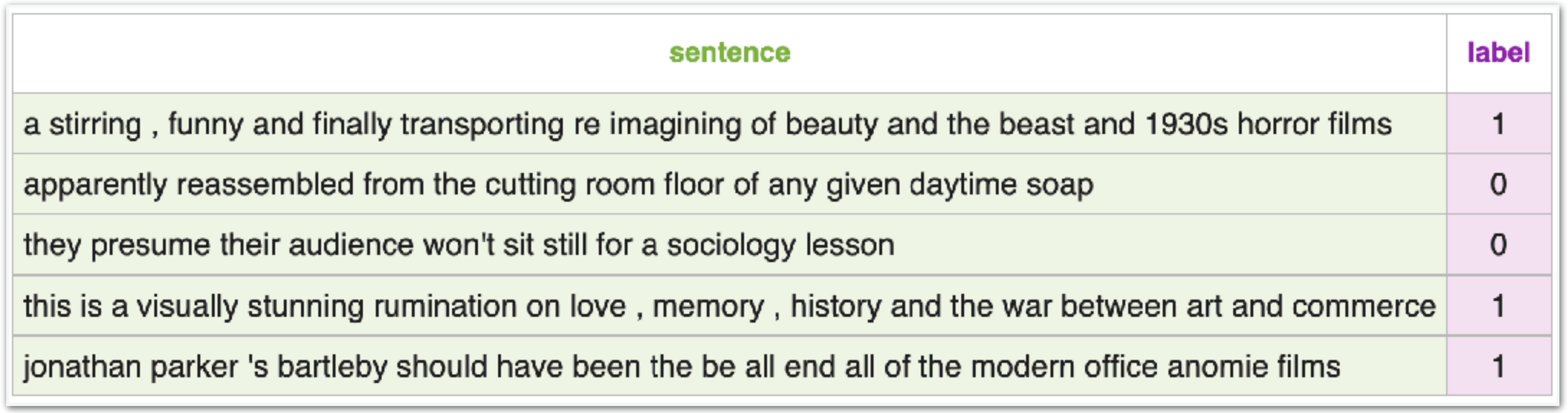
- From attention to self-attention
- From self-attention to multi-head self-attention
- Feedforward layers
- Positional encoding
- Residual connections + layer normalization
- Transformer encoder vs Transformer decoder

## Neural language models: pretraining

# Traditional learning paradigm

- **Supervised training/fine-tuning only, NO pre-training**
  - Collect (x, y) task training pairs

Data:



| sentence | label |
|---|---|
| a stirring , funny and finally transporting re imagining of beauty and the beast and 1930s horror films | 1 |
| apparently reassembled from the cutting room floor of any given daytime soap | 0 |
| they presume their audience won't sit still for a sociology lesson | 0 |
| this is a visually stunning rumination on love , memory , history and the war between art and commerce | 1 |
| jonathan parker 's bartleby should have been the be all end all of the modern office anomie films | 1 |

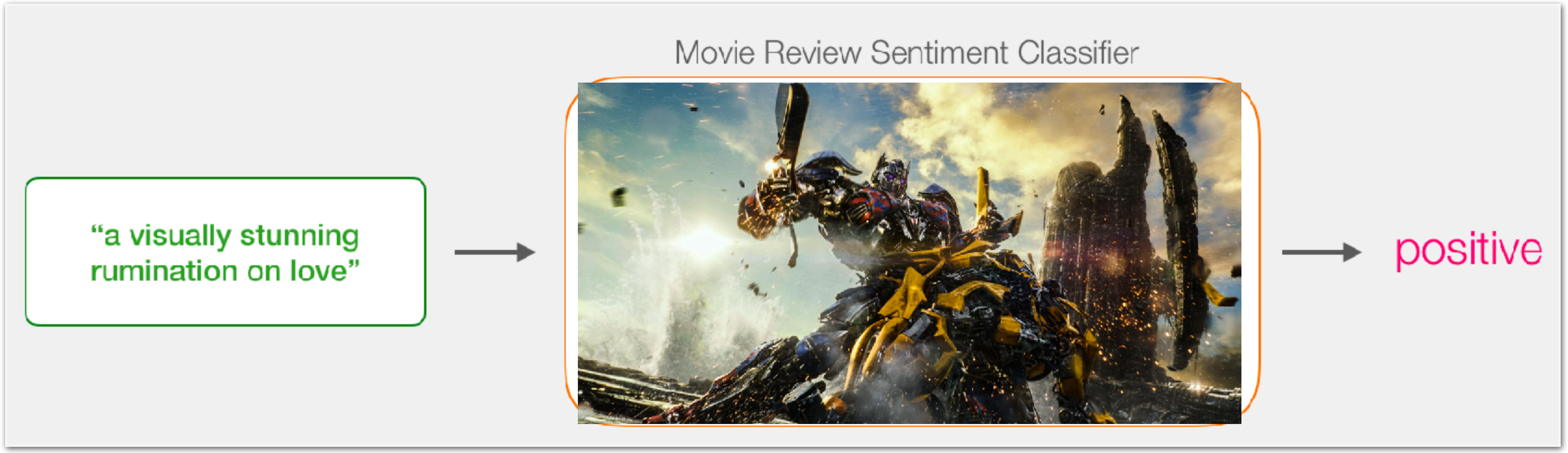
# Traditional learning paradigm

- **Supervised training/fine-tuning only, NO pre-training**
  - Collect (x, y) task training pairs
  - Randomly initialize your models f(x) (e.g., vanilla Transformers)

Randomly initialized Transformers
**NO pretrained** parameters used

# Traditional learning paradigm

- **Supervised training/fine-tuning only, NO pre-training**
  - Collect (x, y) task training pairs
  - Randomly initialize your models f(x) (e.g., vanilla Transformers)
  - Train f(x) on (x, y) pairs


Movie Review Sentiment Classifier
"a visually stunning rumination on love"
positive

Train Transformers only on task labeled data

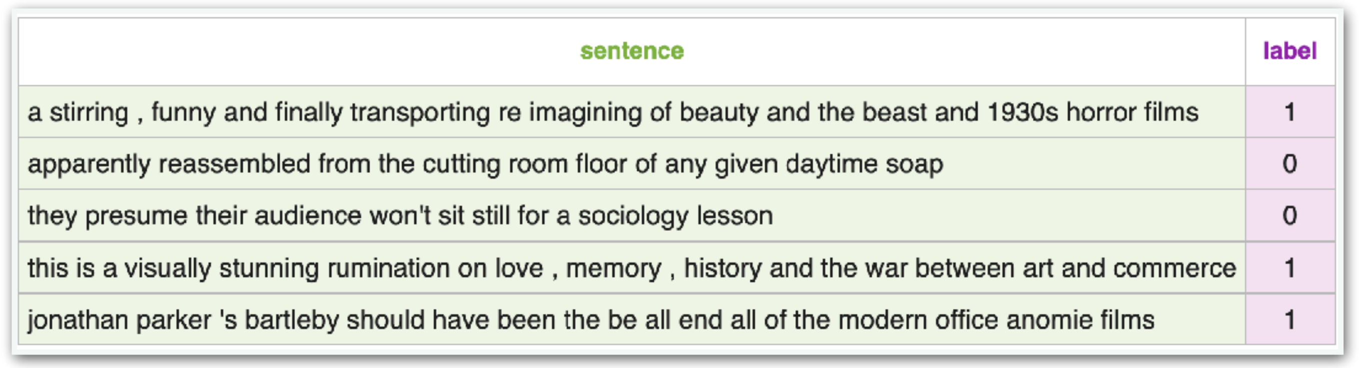
# Traditional learning paradigm

- **Supervised training/fine-tuning only, NO pre-training**
  - Collect (x, y) task training pairs
  - Randomly initialize your models f(x) (e.g., vanilla Transformers)
  - Train f(x) on (x, y) pairs

Then you get a trained Transformers **ONLY** for sentiment analysis
The model can be: NB, LR, RNNs, LSTM too

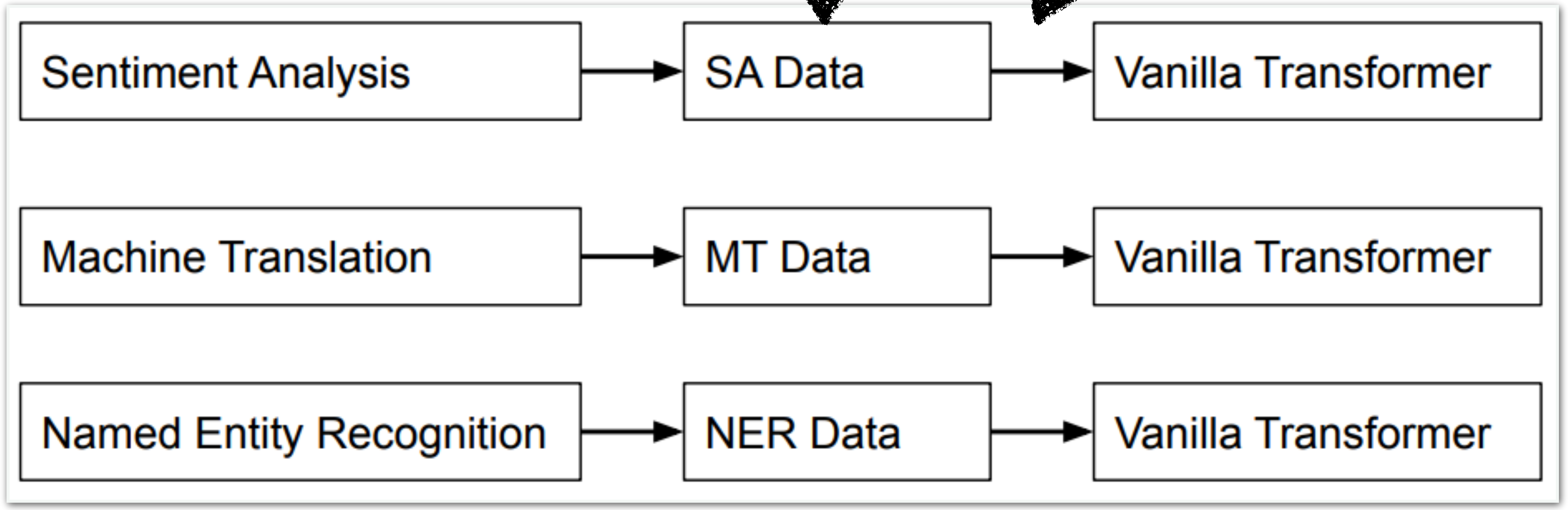
## Traditional learning paradigm

- **Supervised training/fine-tuning only, NO pre-training**
  - Train Transformer or other models separately for each task



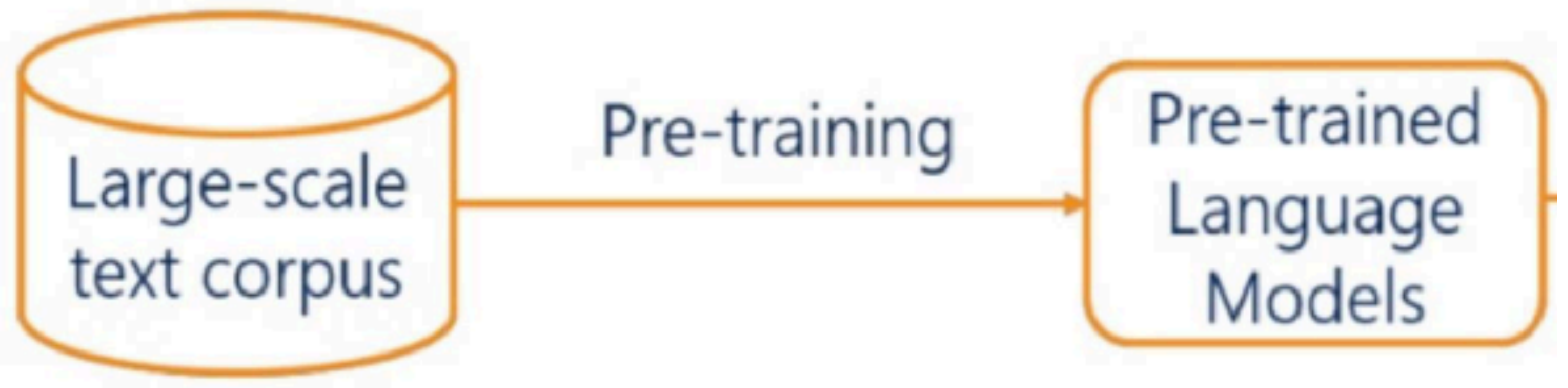
| sentence | label |
|---|---|
| a stirring , funny and finally transporting re imagining of beauty and the beast and 1930s horror films | 1 |
| apparently reassembled from the cutting room floor of any given daytime soap | 0 |
| they presume their audience won't sit still for a sociology lesson | 0 |
| this is a visually stunning rumination on love , memory , history and the war between art and commerce | 1 |
| jonathan parker 's bartleby should have been the be all end all of the modern office anomie films | 1 |

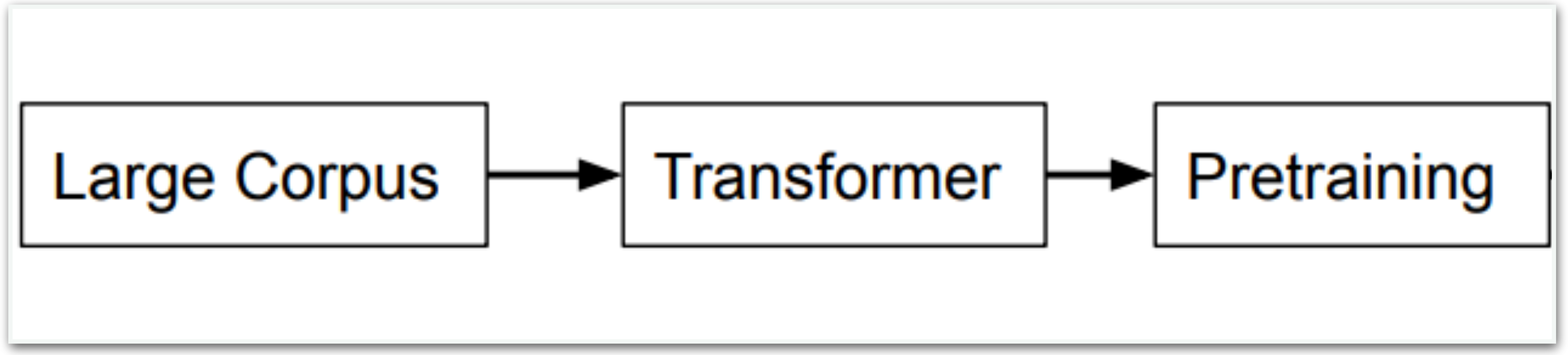
Supervised fine-tuning only
NO pre-training



# Modern learning paradigm

- **Pre-training + supervised training/fine-tuning**
  - First train Transformer using a lot of general text using unsupervised learning. This is called **pretraining**.

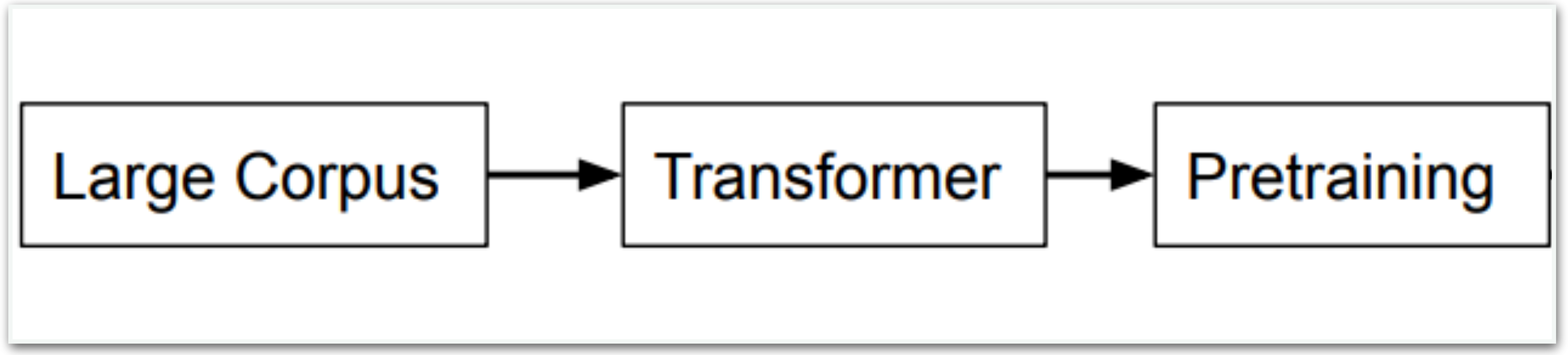




## Modern learning paradigm

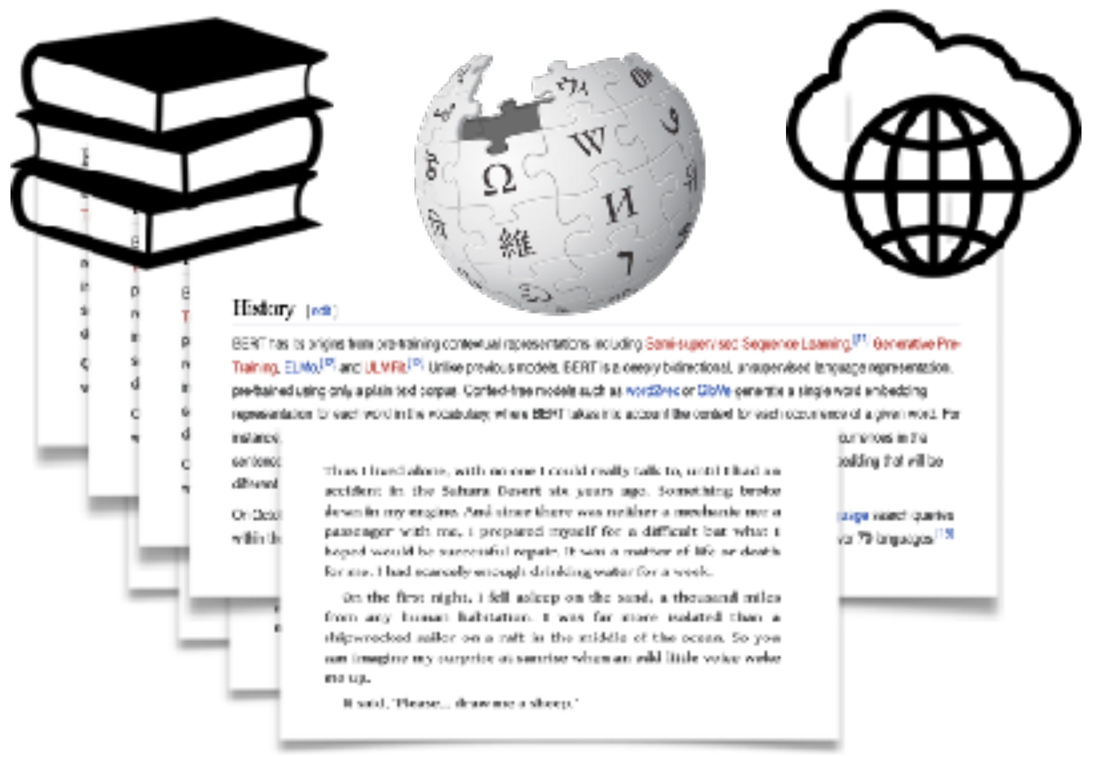
- **Pre-training + supervised training/fine-tuning**
  - First train Transformer using a lot of general text using unsupervised learning. This is called **pretraining**.

1. Randomly initialized Transformers



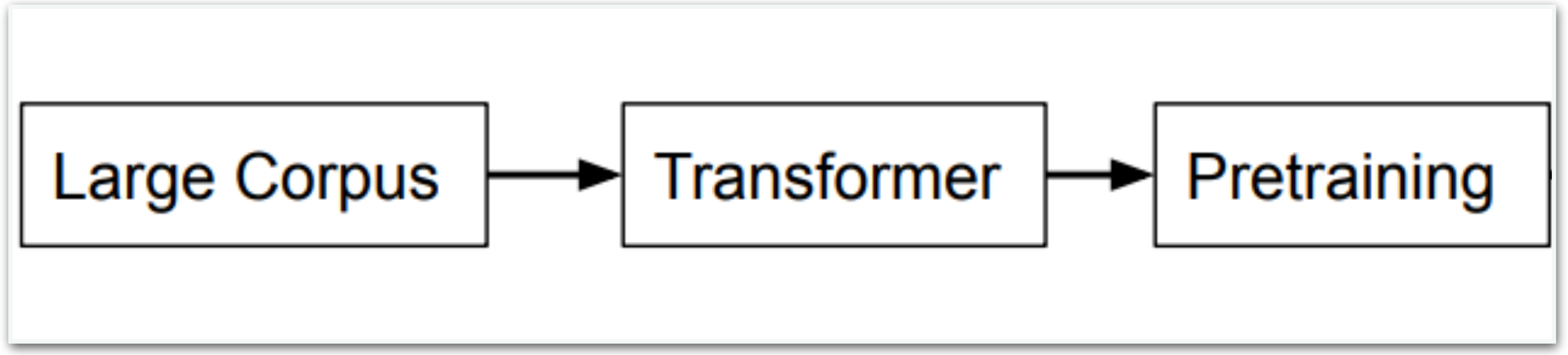
# Modern learning paradigm

- **Pre-training + supervised training/fine-tuning**
  - First train Transformer using a lot of general text using unsupervised learning. This is called **pretraining**.



**general training objectives?**

2. Pre-train Transformers on large text data with **general objectives**

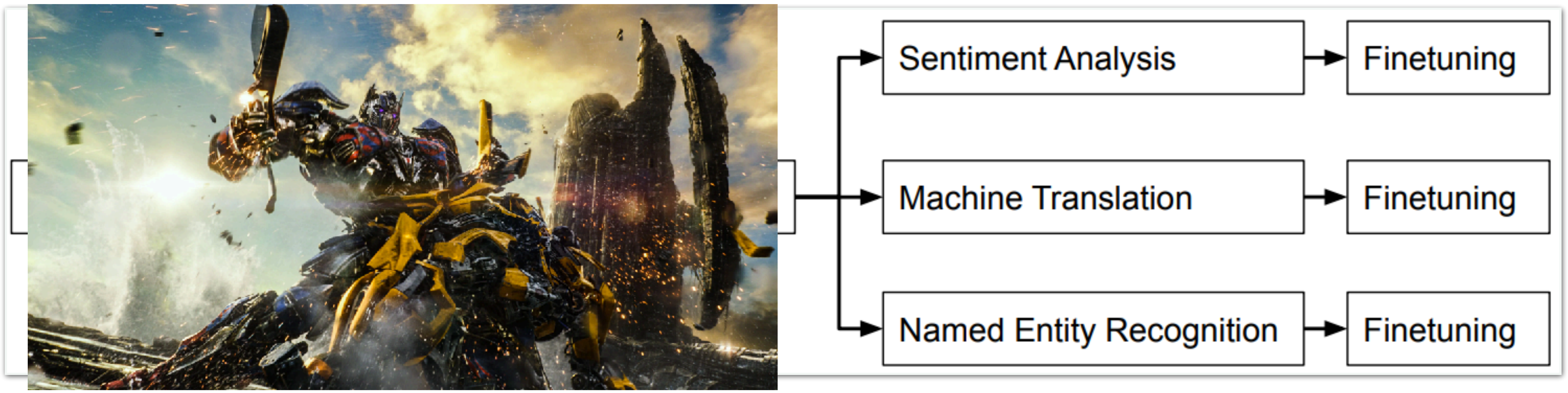


>2M GPU hours

# Modern learning paradigm

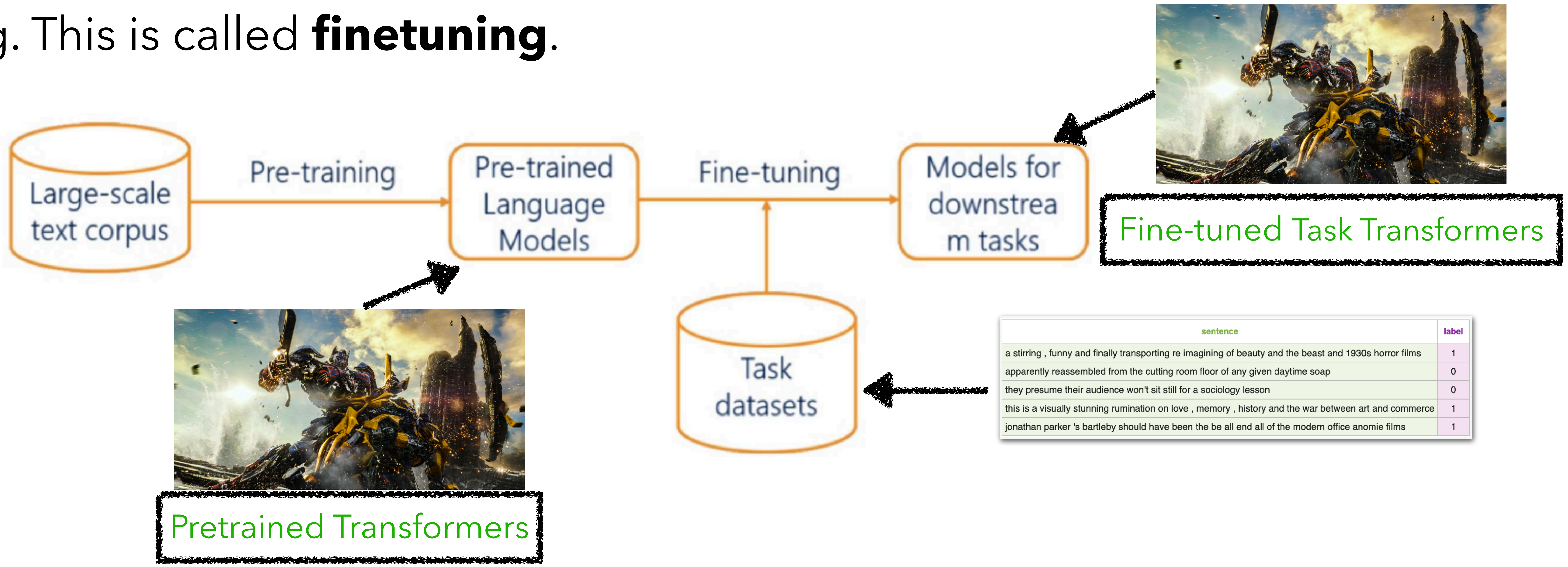
- **Pre-training + supervised training/fine-tuning**
  - First train Transformer using a lot of general text using unsupervised learning. This is called **pretraining**.
  - Then train the pretrained Transformer for a specific task using supervised learning. This is called **finetuning**.

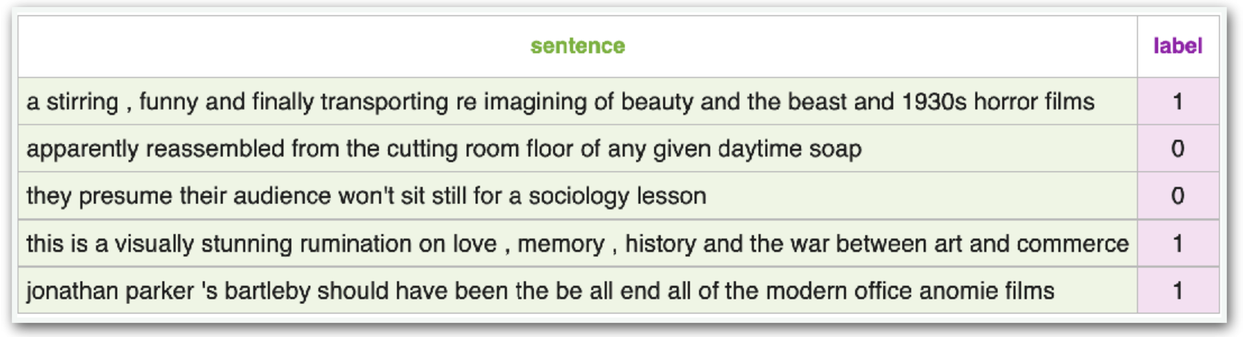
Pretrained Transformers



# Modern learning paradigm

- **Pre-training + supervised training/fine-tuning**
  - First train Transformer using a lot of general text using unsupervised learning. This is called **pretraining**.
  - Then train the pretrained Transformer for a specific task using supervised learning. This is called **finetuning**.


Large-scale text corpus
Pre-training
Pre-trained Language Models
Fine-tuning
Models for downstrea m tasks
Fine-tuned Task Transformers
Task datasets
Pretrained Transformers



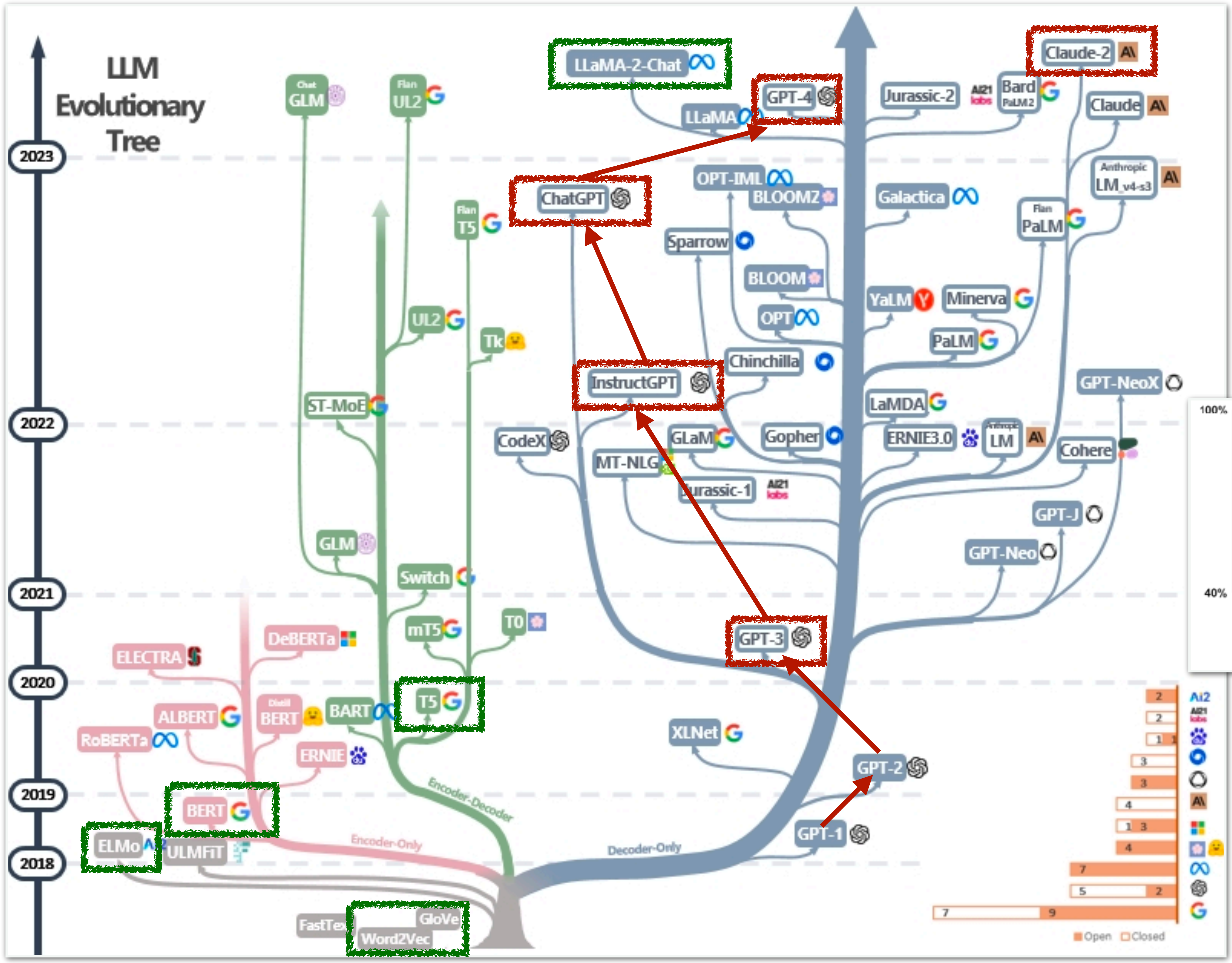
| sentence | label |
|---|---|
| a stirring , funny and finally transporting re imagining of beauty and the beast and 1930s horror films | 1 |
| apparently reassembled from the cutting room floor of any given daytime soap | 0 |
| they presume their audience won't sit still for a sociology lesson | 0 |
| this is a visually stunning rumination on love , memory , history and the war between art and commerce | 1 |
| jonathan parker 's bartleby should have been the be all end all of the modern office anomie films | 1 |

# Evolution tree of pretrained LMs

~200 billion

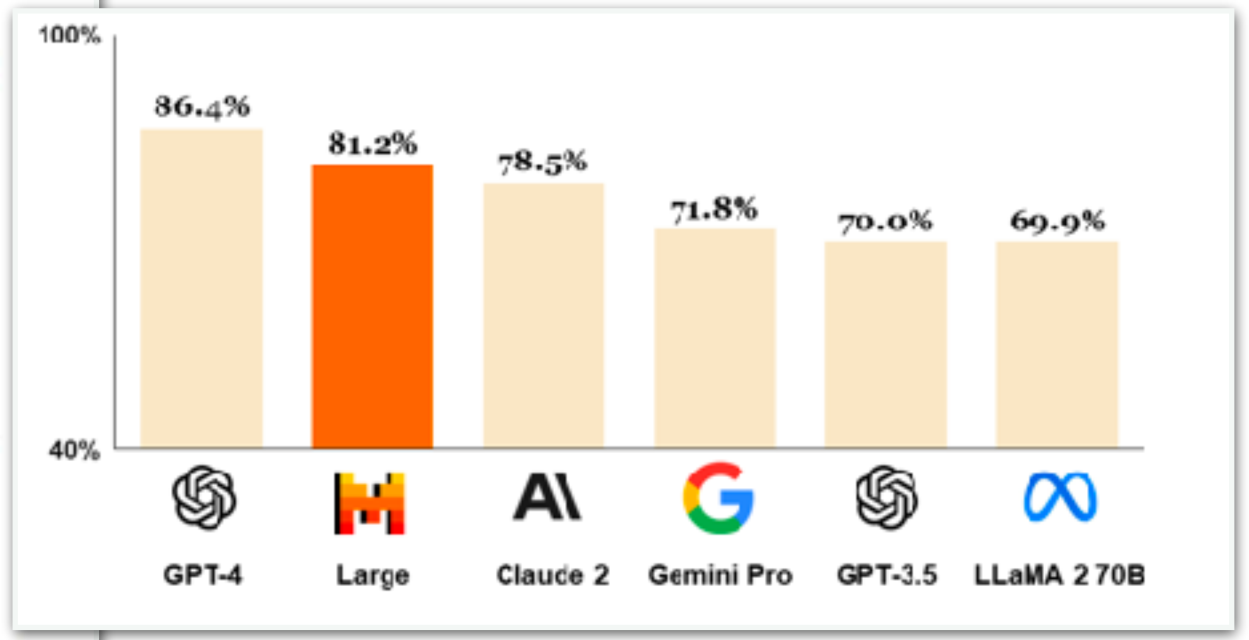
Model size
(# of parameters)
~1000 times larger

~300 million



🟩 Open-sourced

🟥 Close-sourced



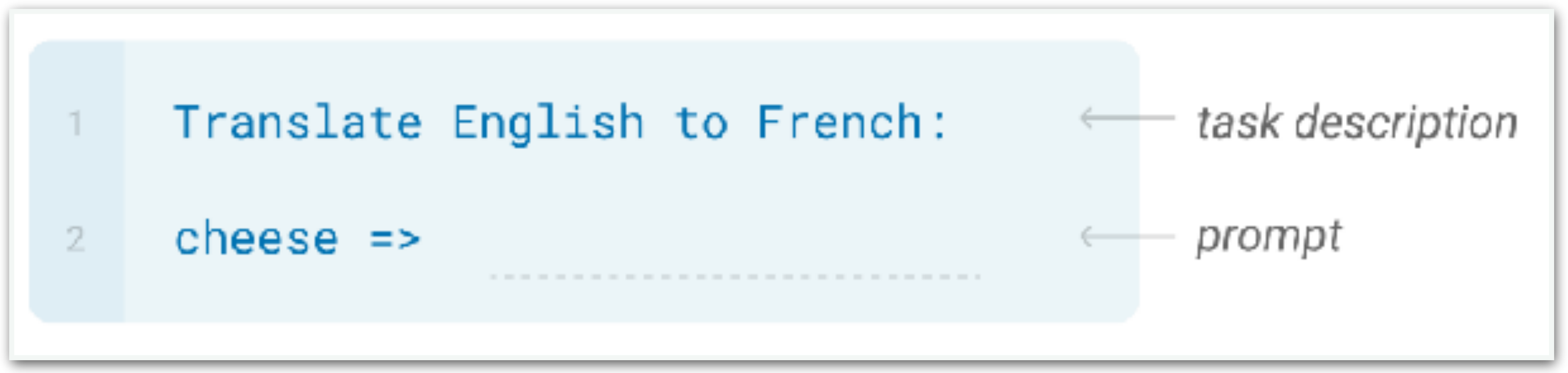
https://github.com/Mooler0410/LLMsPracticalGuide
https://mistral.ai/news/mistral-large/

## Latest learning paradigm with LLMs

- **Pre-training + prompting/in-context learning (no training this step)**
    - First train a **large (>7~175B)** Transformer using a lot of general text using unsupervised learning. This is called **large** language model **pretraining**.

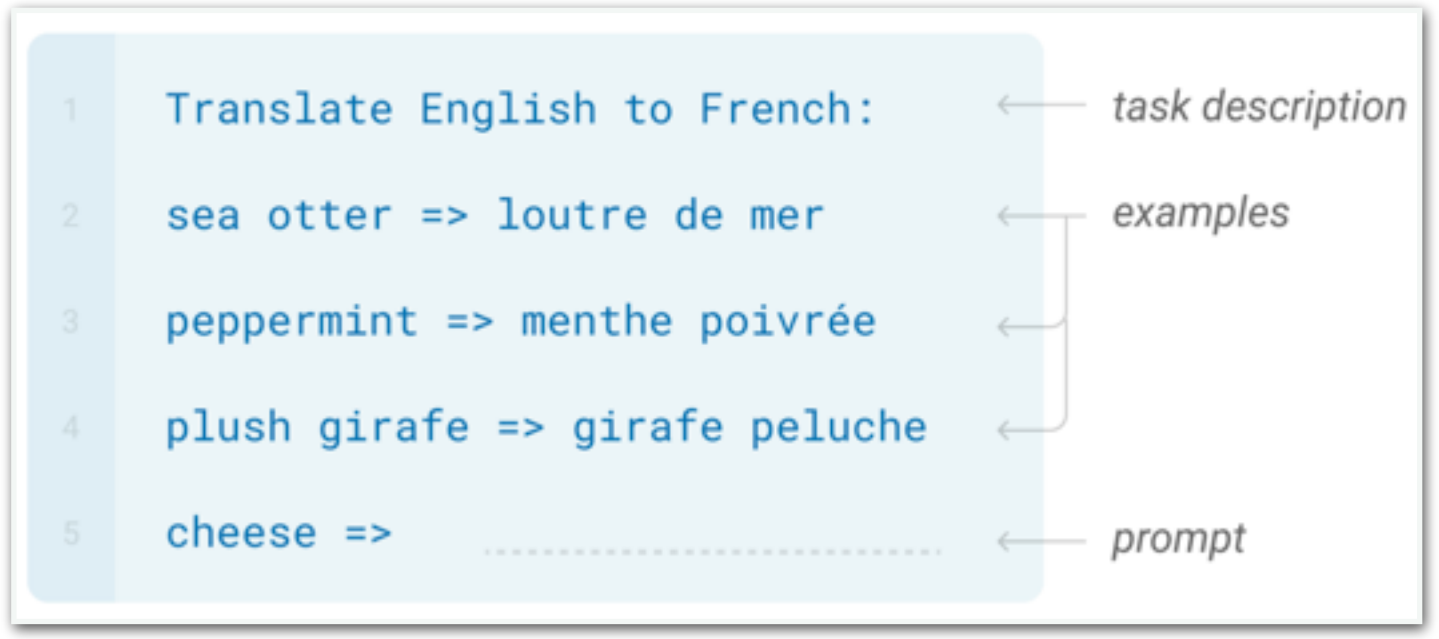
# Latest learning paradigm with LLMs

- **Pre-training + prompting/in-context learning (no training this step)**
  - First train a **large (>7~175B)** Transformer using a lot of general text using unsupervised learning. This is called **large** language model **pretraining**.
  - Then **directly use** the pretrained large Transformer (**no further finetuning/ training**) for any different task given only a natural language description of the task or a few task (x, y) examples. This is called **prompting/in-context learning**.



```
Translate English to French:
cheese =>
```

Zero-shot prompting

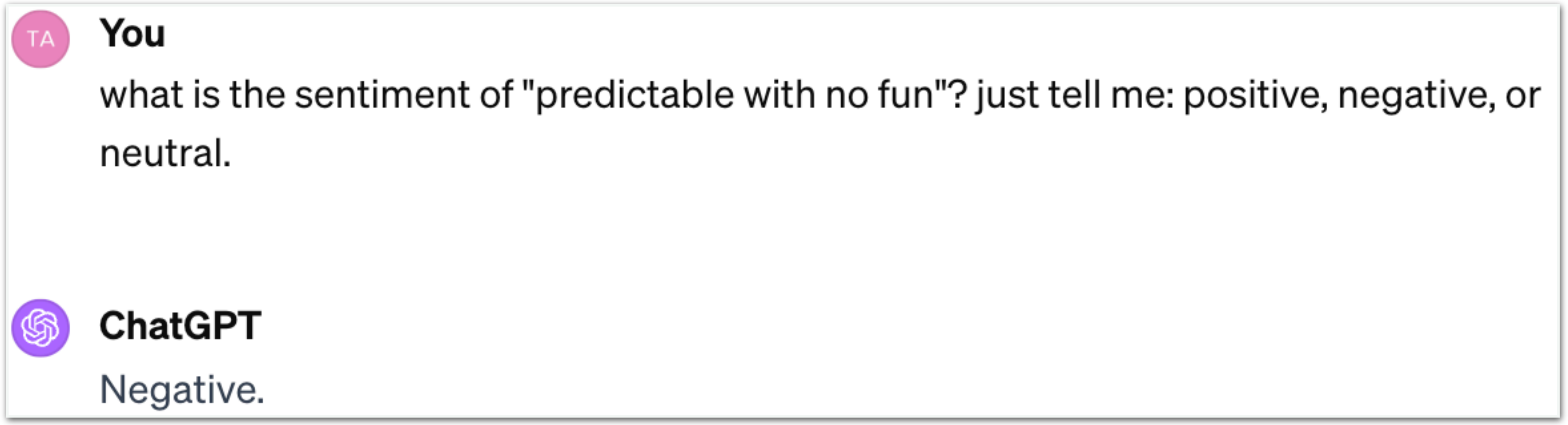


```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

Few-shot prompting/in-context learning

# Example: Prompting ChatGPT for sentiment analysis

- **Pre-training + prompting/in-context learning (no training this step)**



TA **You**

what is the sentiment of "predictable with no fun"? just tell me: positive, negative, or neutral.

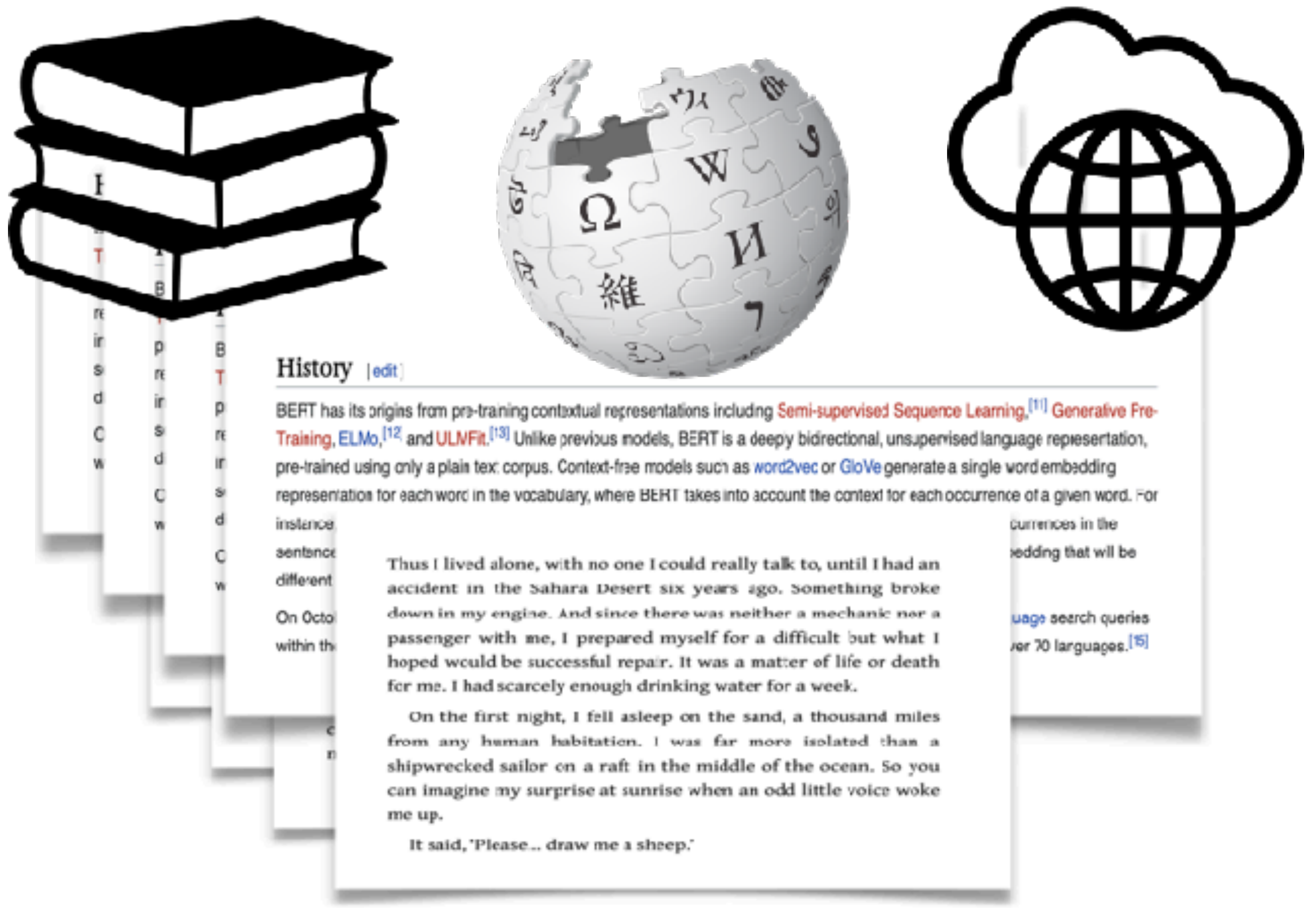
**ChatGPT**

Negative.

Already pretrained ChatGPT
No further training for sentiment analysis
Just prompting to conduct the task!

# Pretraining: training objectives?

- During pretraining, we have a large text corpus (**no task labels**)
  - **Key question: what labels or objectives used to train the vanilla Transformers?**



History [edit]

BERT has its origins from pre-training contextual representations including Semi-supervised Sequence Learning,[11] Generative Pre-Training, ELMo,[12] and ULMFit.[13] Unlike previous models, BERT is a deeply bidirectional, unsupervised language representation, pre-trained using only a plain text corpus. Context-free models such as word2vec or GloVe generate a single word embedding representation for each word in the vocabulary, where BERT takes into account the context for each occurrence of a given word. For instance ... currences in the sentence ... edding that will be different

On Octo ... uage search queries within th ... er 70 languages.[15]

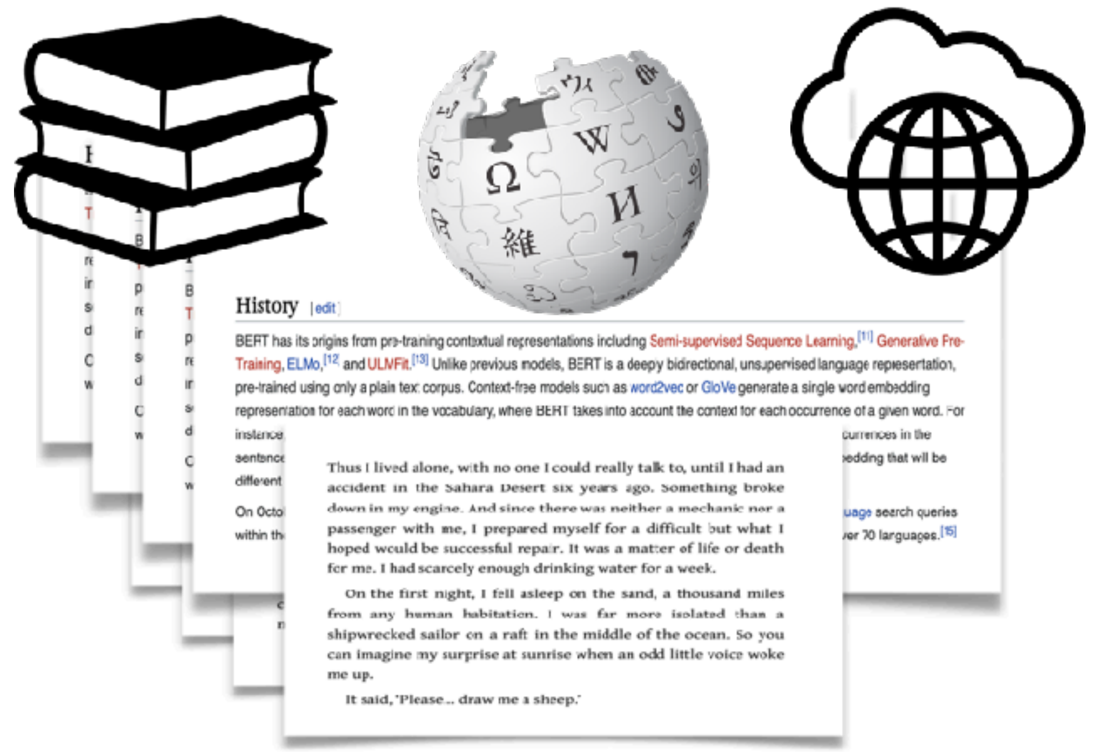
Thus I lived alone, with no one I could really talk to, until I had an accident in the Sahara Desert six years ago. Something broke down in my engine. And since there was neither a mechanic nor a passenger with me, I prepared myself for a difficult but what I hoped would be successful repair. It was a matter of life or death for me. I had scarcely enough drinking water for a week.

On the first night, I fell asleep on the sand, a thousand miles from any human habitation. I was far more isolated than a shipwrecked sailor on a raft in the middle of the ocean. So you can imagine my surprise at sunrise when an odd little voice woke me up.

It said, 'Please... draw me a sheep.'

# Pretraining: training objectives?

- During pretraining, we have a large text corpus (**no task labels**)
  - **Key question: what labels or objectives used to train the vanilla Transformers?**



→

→

**Training labels/objectives?**

# Pretraining: training objectives?

BERT

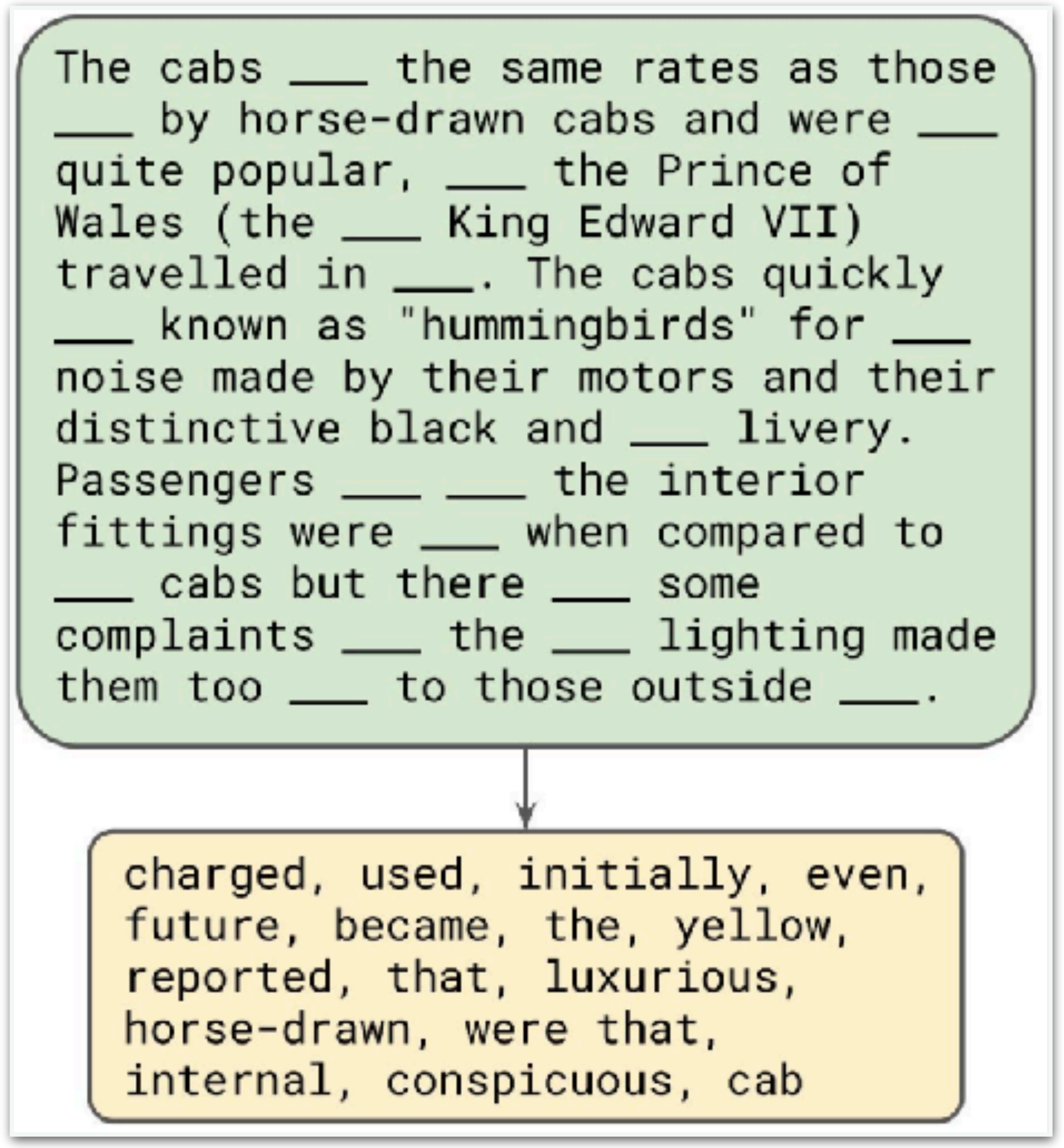
Devlin et al., 2018

T5

Raffel et al., 2019

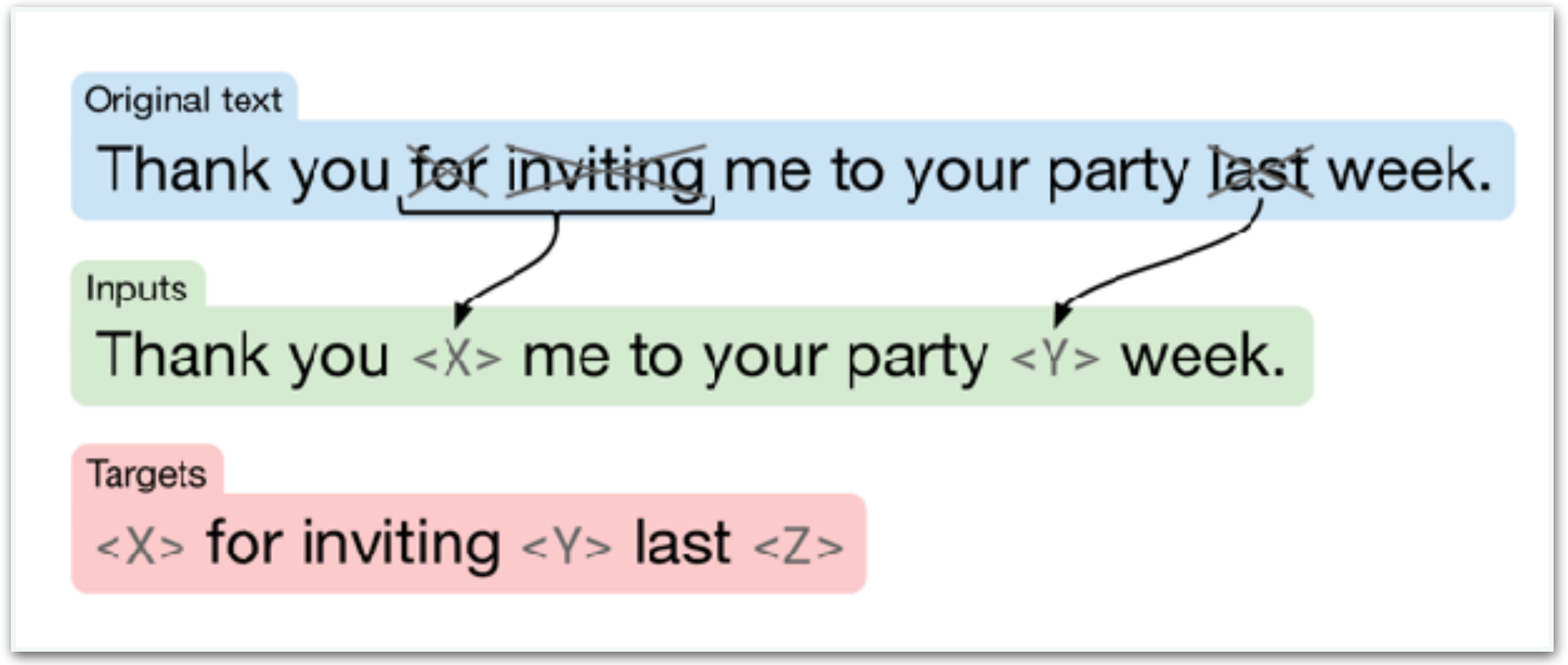
OpenAI

GPT - 4



The cabs ___ the same rates as those ___ by horse-drawn cabs and were ___ quite popular, ___ the Prince of Wales (the ___ King Edward VII) travelled in ___. The cabs quickly ___ known as "hummingbirds" for ___ noise made by their motors and their distinctive black and ___ livery. Passengers ___ ___ the interior fittings were ___ when compared to ___ cabs but there ___ some complaints ___ the ___ lighting made them too ___ to those outside ___.

charged, used, initially, even, future, became, the, yellow, reported, that, luxurious, horse-drawn, were that, internal, conspicuous, cab



Original text

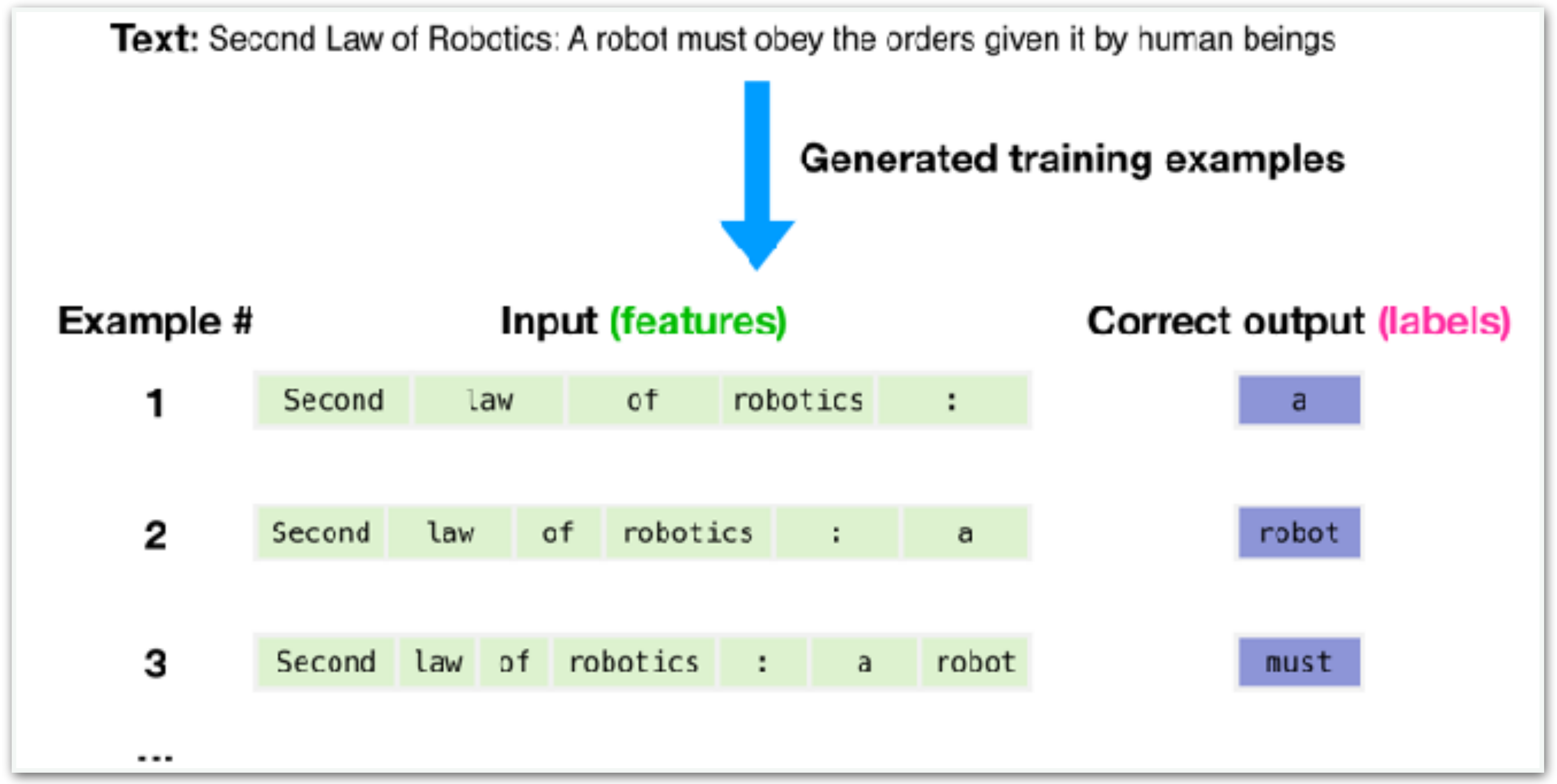
Thank you ~~for inviting~~ me to your party ~~last~~ week.

Inputs

Thank you <X> me to your party <Y> week.

Targets

<X> for inviting <Y> last <Z>



**Text:** Second Law of Robotics: A robot must obey the orders given it by human beings

**Generated training examples**

| Example # | Input (features) | Correct output (labels) |
|---|---|---|
| 1 | Second law of robotics : | a |
| 2 | Second law of robotics : a | robot |
| 3 | Second law of robotics : a robot | must |
| ... | | |

Masked token prediction

Denoising span-mask prediction

Next token prediction

https://github.com/manueldeprada/Pretraining-T5-PyTorch-Lightning

## Advantages of pre-training

- **Leveraging rich underlying information** from abundant raw texts.
- **Reducing the reliance of task-specific labeled data** that is difficult or costly to obtain.
- **Initializing model parameters** for more **generalizable** NLP applications.
- **Saving training cost** by providing a reusable model checkpoints.
- **Providing robust representation** of language contexts.

# Pre-training architectures

**Encoder**

- E.g., BERT, RoBERTa, DeBERTa, …
- **Autoencoder** model
- **Masked** language modeling

**Encoder-Decoder**

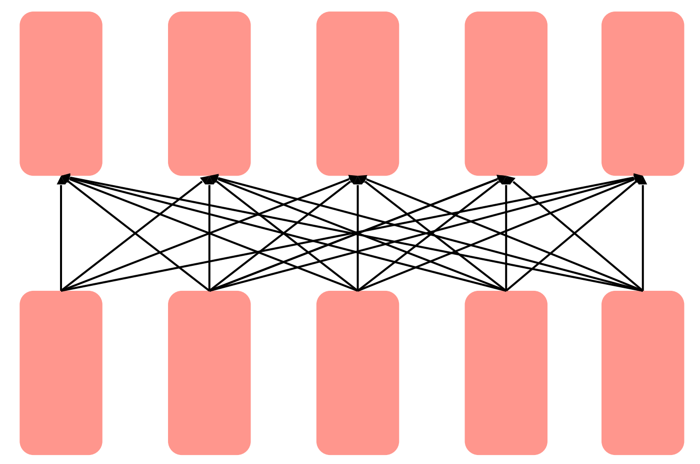
- E.g., T5, BART, …
- **seq2seq** model

**Decoder**

- E.g., GPT, GPT2, GPT3, …
- **Autoregressive** model
- **Left-to-right** language modeling

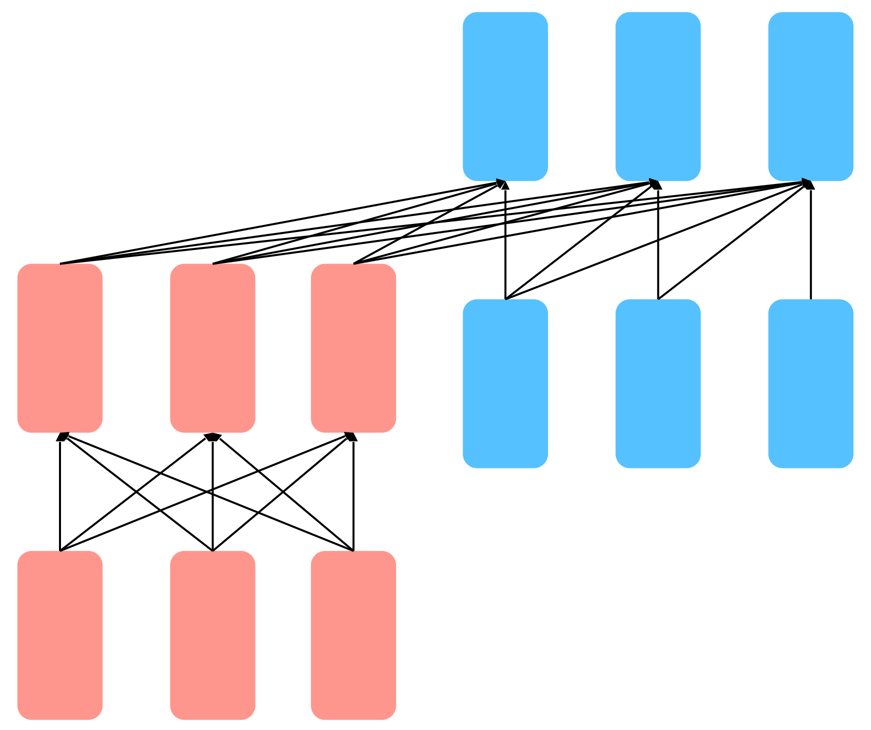
# Pre-training architectures

**Encoder**



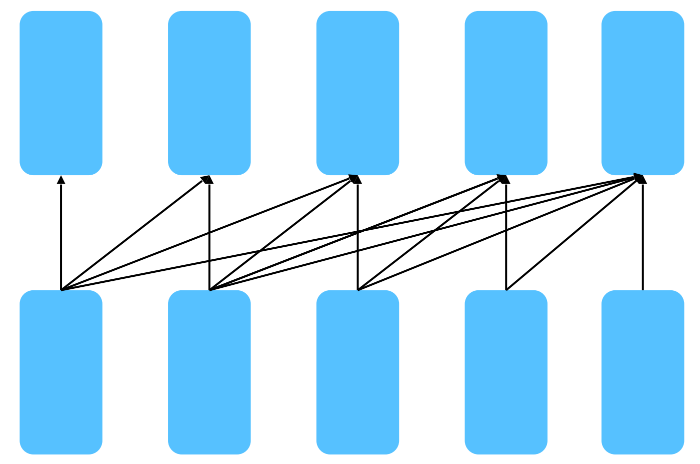
- Bidirectional; can condition on the future context

**Encoder-Decoder**



- Map two sequences of different length together

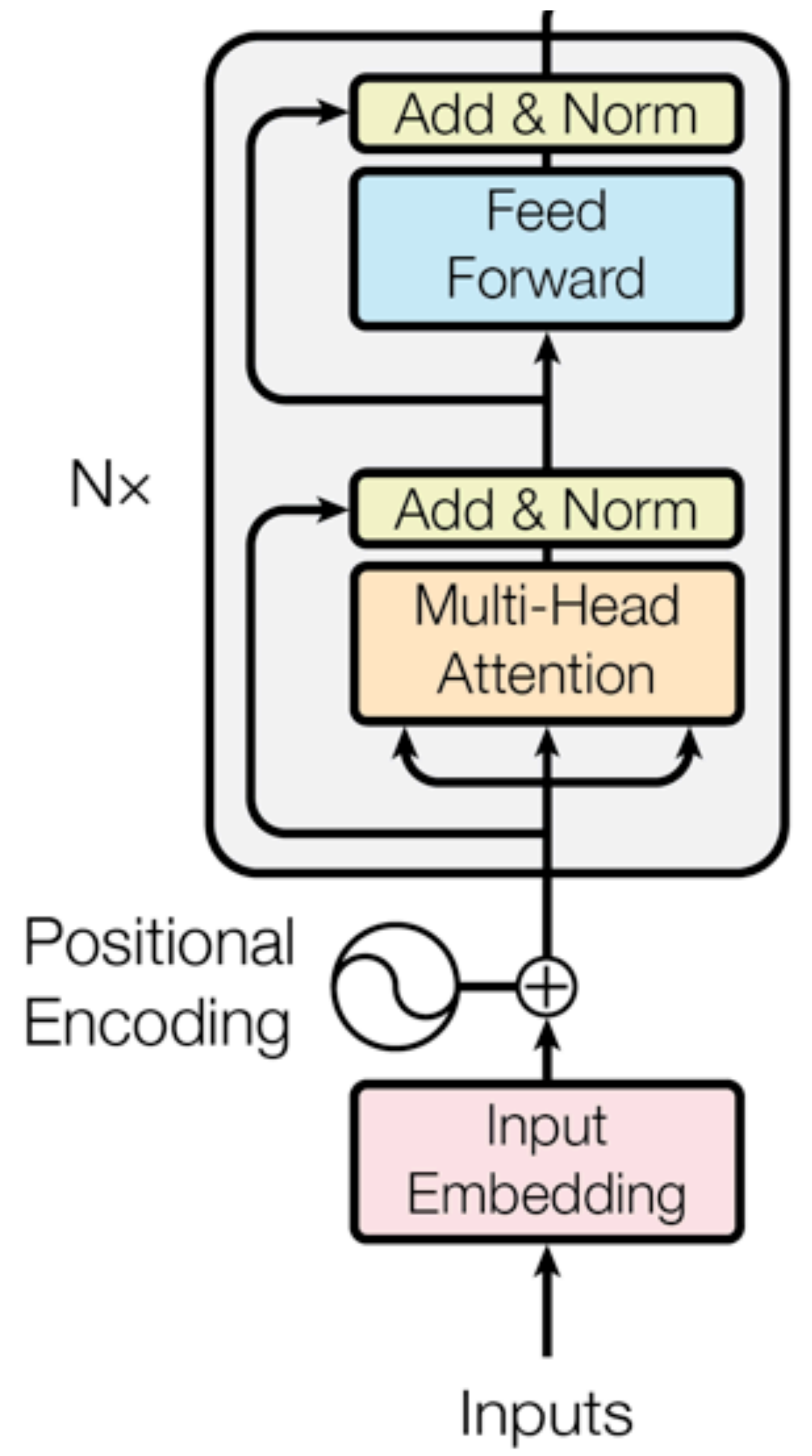
**Decoder**



- Language modeling; can only condition on the past context

# BERT: Bidirectional Encoder Representations from Transformers

(Released in 2018/10)



- It is a fine-tuning approach based on a deep **bidirectional Transformer encoder** instead of a Transformer decoder
- The key: learn representations based on **bidirectional contexts**

Example #1: we went to the river bank.

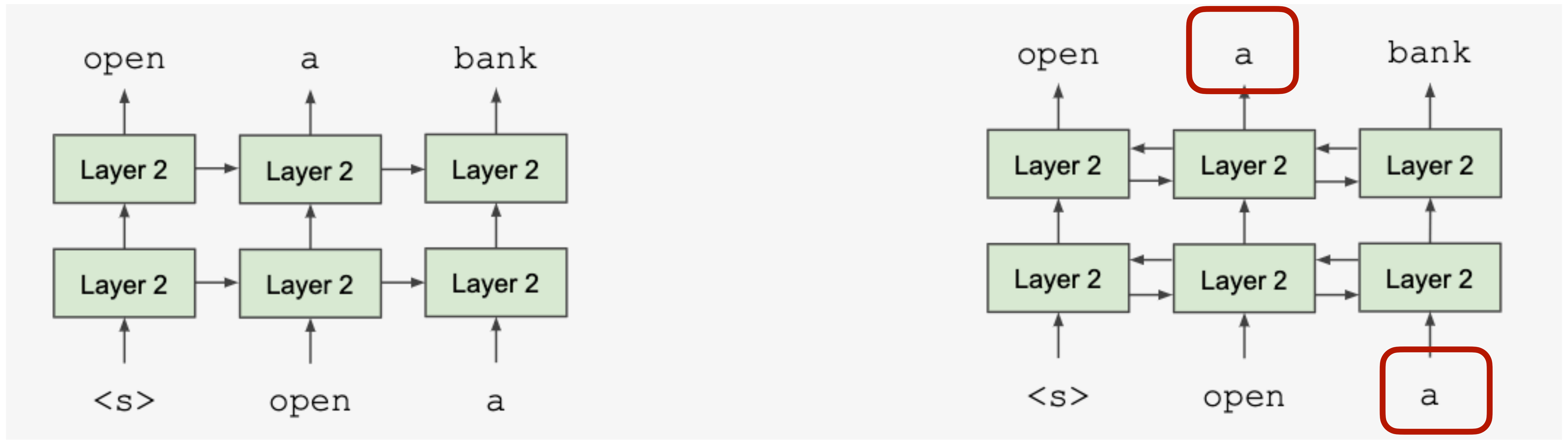
Example #2: I need to go to bank to make a deposit.

- Two new pre-training objectives:
  - **Masked language modeling (MLM)**
  - Next sentence prediction (NSP) - Later work shows that NSP hurts performance though..

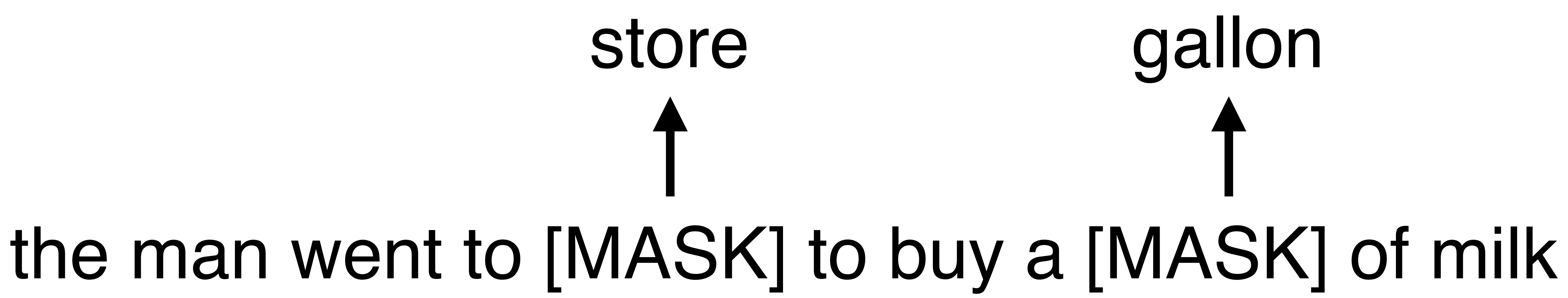
(Devlin et al, 2019): BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

# Masked Language Modeling (MLM)

- Q: Why we can't do language modeling with bidirectional models?

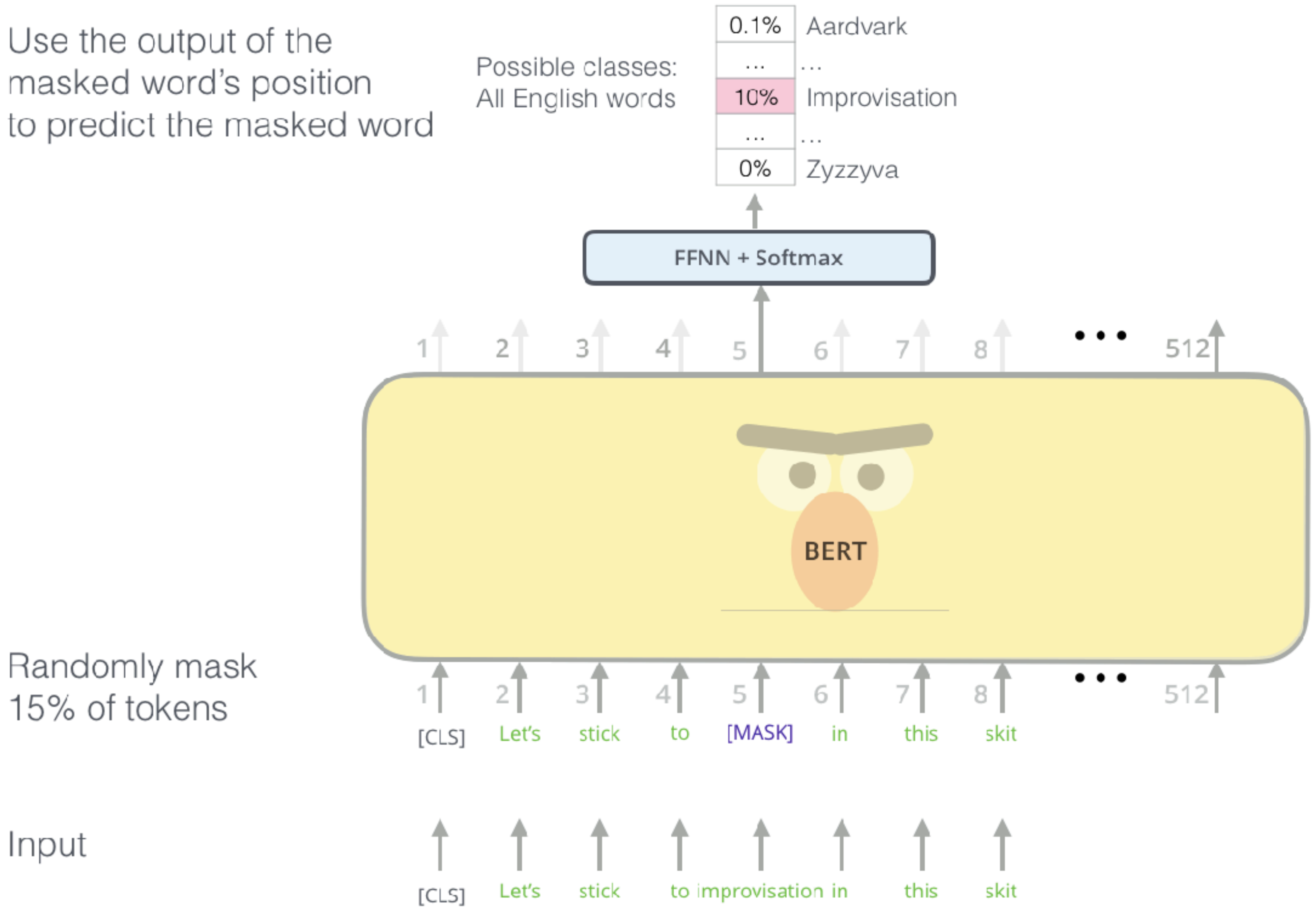


- Solution: Mask out k% of the input words, and then predict the masked words



k = 15% in practice

## Masked Language Modeling (MLM)



Use the output of the masked word's position to predict the masked word
Possible classes: All English words
0.1% Aardvark
... ...
10% Improvisation
... ...
0% Zyzzyva
FFNN + Softmax
1 2 3 4 5 6 7 8 ... 512
BERT
Randomly mask 15% of tokens
[CLS] Let's stick to [MASK] in this skit
Input
[CLS] Let's stick to improvisation in this skit

http://jalammar.github.io/illustrated-bert/

## MLM: 80-10-10 corruption

For the 15% predicted words,

- 80% of the time, they replace it with [MASK] token

went to the store $\longrightarrow$ went to the [MASK]

- 10% of the time, they replace it with a random word in the vocabulary

went to the store $\longrightarrow$ went to the running

- 10% of the time, they keep it unchanged

went to the store $\longrightarrow$ went to the store

Why? Because [MASK] tokens are never seen during fine-tuning

(See Table 8 of the paper for an ablation study)

## Next Sentence Prediction (NSP)

- Motivation: many NLP downstream tasks require understanding the relationship between two sentences (natural language inference, paraphrase detection, QA)
- NSP is designed to reduce the gap between pre-training and fine-tuning

[CLS]: a special token always at the beginning

[SEP]: a special token used to separate two segments

Input = `[CLS] the man went to [MASK] store [SEP]`
`he bought a gallon [MASK] milk [SEP]`

Label = `IsNext`

Input = `[CLS] the man [MASK] to the store [SEP]`
`penguin [MASK] are flight ##less birds [SEP]`

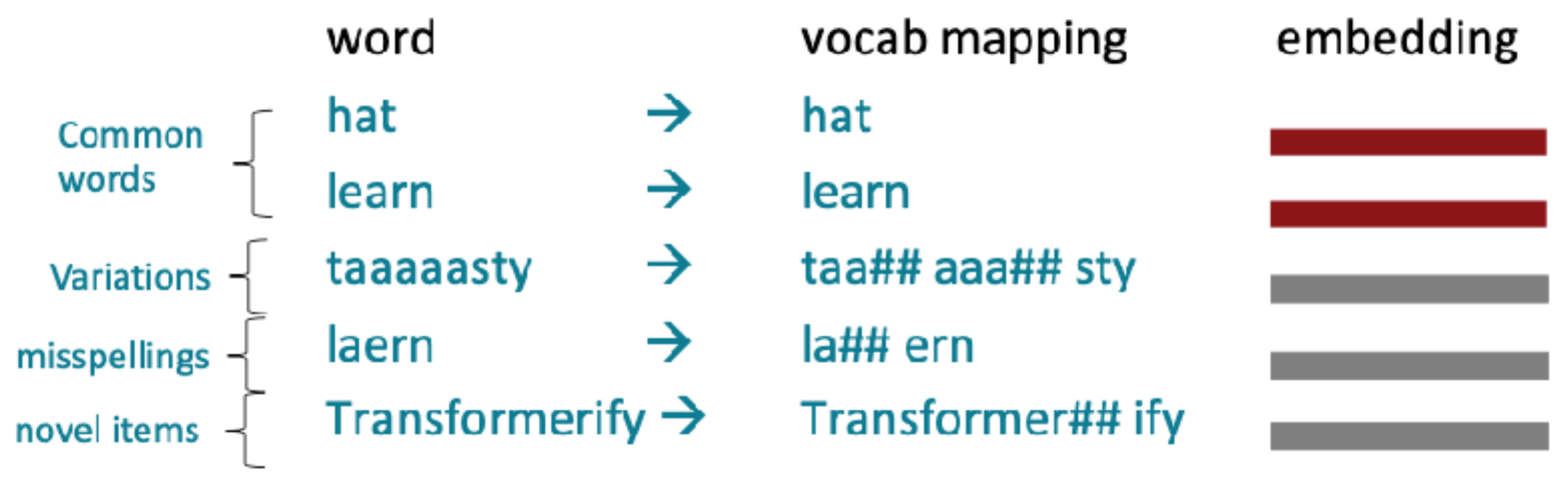
Label = `NotNext`

They sample two contiguous segments for 50% of the time and another random segment from the corpus for 50% of the time

This actually hurts model learning based on later work!

# BERT pre-training

- Vocabulary size: 30,000 wordpieces (common sub-word units) (Wu et al., 2016)

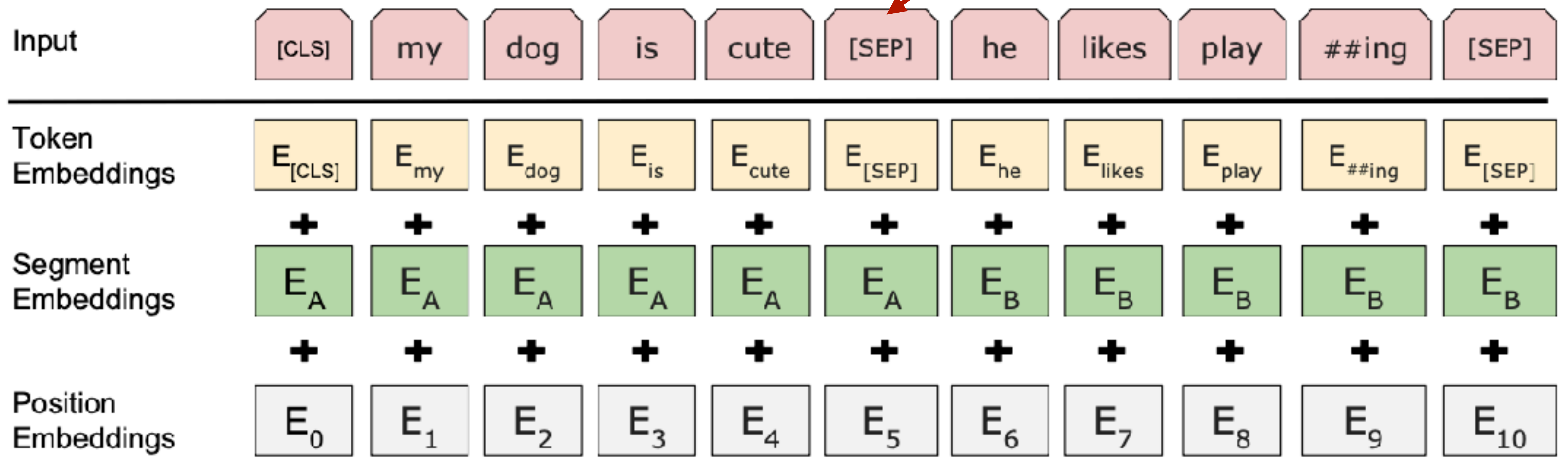


(Image: Stanford CS224N)

- Input embeddings:

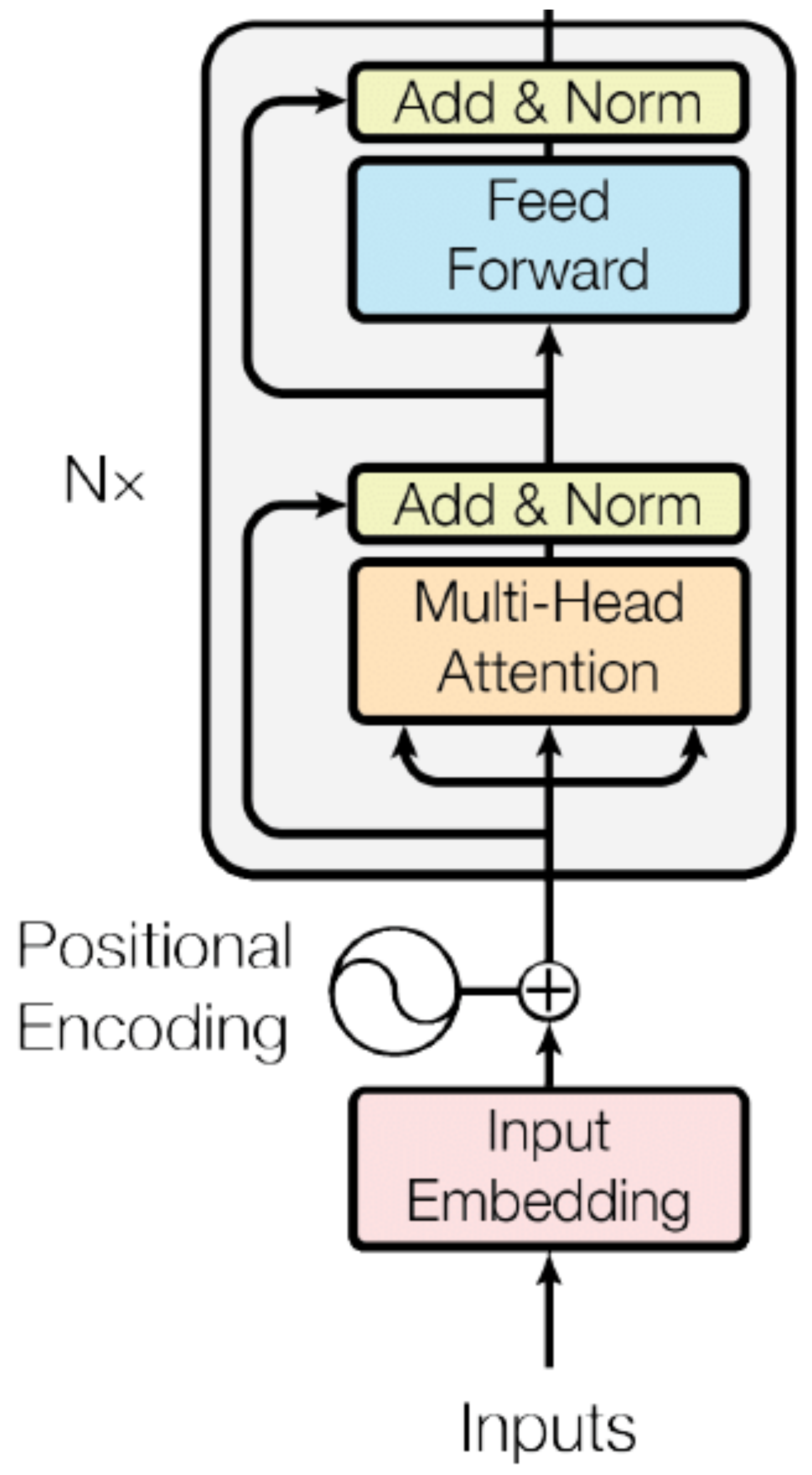
Special token added to the beginning of each input sequence

Special token to separate sentence A/B


Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ $E_{my}$ $E_{dog}$ $E_{is}$ $E_{cute}$ $E_{[SEP]}$ $E_{he}$ $E_{likes}$ $E_{play}$ $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
$E_A$ $E_A$ $E_A$ $E_A$ $E_A$ $E_A$ $E_B$ $E_B$ $E_B$ $E_B$ $E_B$
Position Embeddings
$E_0$ $E_1$ $E_2$ $E_3$ $E_4$ $E_5$ $E_6$ $E_7$ $E_8$ $E_9$ $E_{10}$

Separate two segments

# BERT pre-training



- BERT-base: 12 layers, 768 hidden size, 12 attention heads, 110M parameters
- BERT-large: 24 layers, 1024 hidden size, 16 attention heads, 340M parameters

- Training corpus: Wikipedia (2.5B) + BooksCorpus (0.8B)
- Max sequence size: 512 wordpiece tokens (roughly 256 and 256 for two non-contiguous sequences)
- Trained for 1M steps, batch size 128k

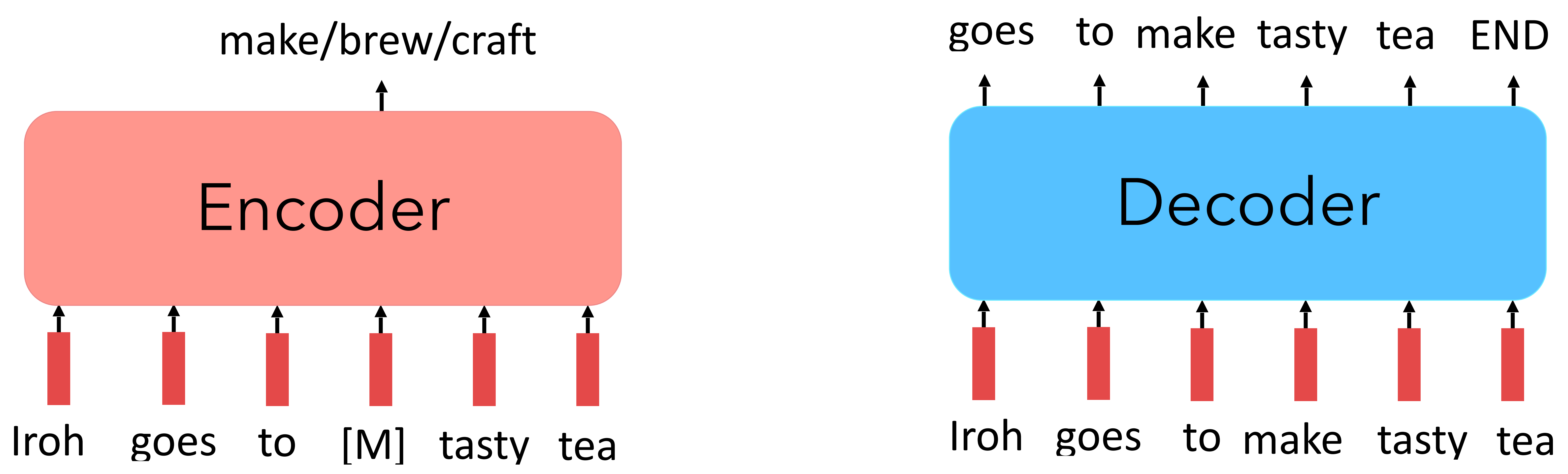
# Encoder: other variations of BERT

- **ALBERT [Lan et al., 2020]**: incorporates two parameter reduction techniques that lift the major obstacles in scaling pre-trained models
- **DeBERTa [He et al., 2021]:** decoding-enhanced BERT with disentangled attention
- **SpanBERT [Joshi et al., 2019]:** masking contiguous spans of words makes a harder, more useful pre-training task
- **ELECTRA [Clark et al., 2020]:** corrupts texts by replacing some tokens with plausible alternatives sampled from a small generator network, then train a discriminative model that predicts whether each token in the corrupted input was replaced by a generator sample or not.
- **DistilBERT [Sanh et al., 2019]:** distilled version of BERT that's 40% smaller
- **TinyBERT [Jiao et al., 2019]:** distill BERT for both pre-training & fine-tuning
- **…**

# Encoder: pros & cons

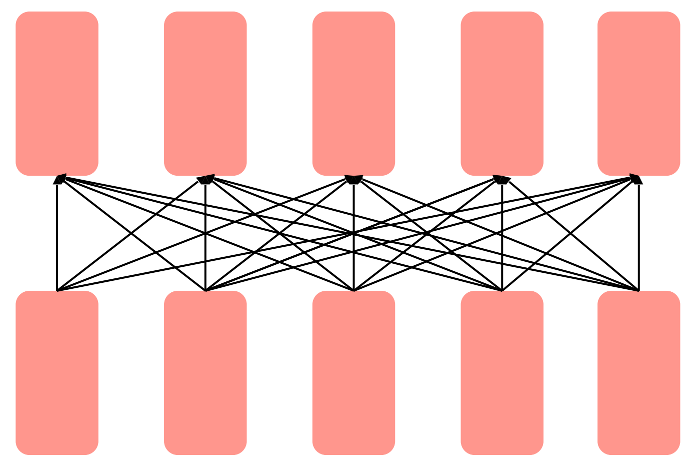
- Consider both left and right context
- Capture intricate contextual relationships

- Not good at generating open-text from left-to-right, one token at a time



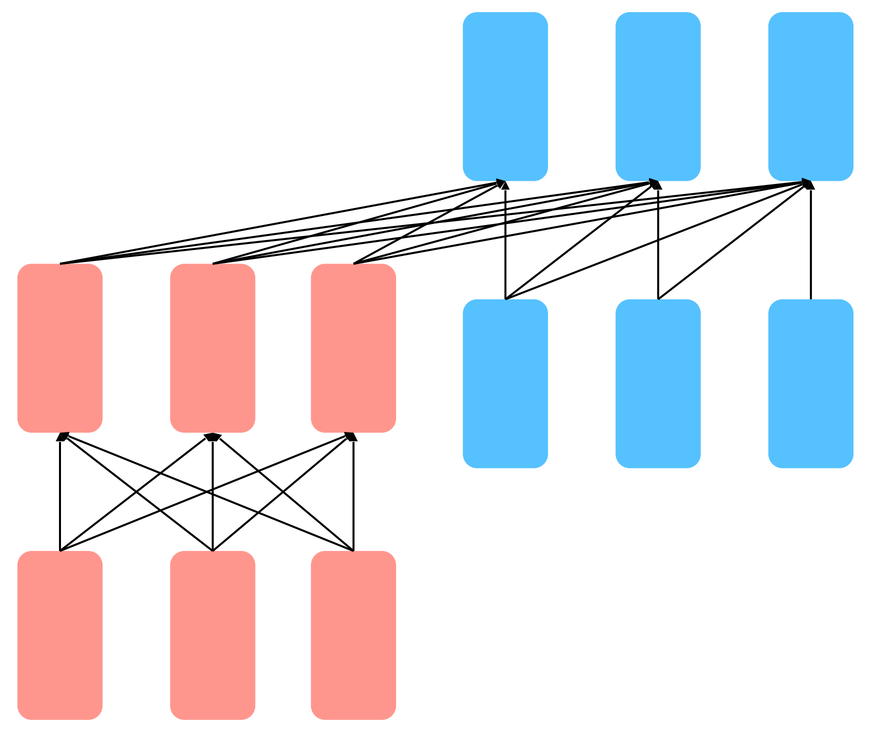
# Pre-training architectures

**Encoder**



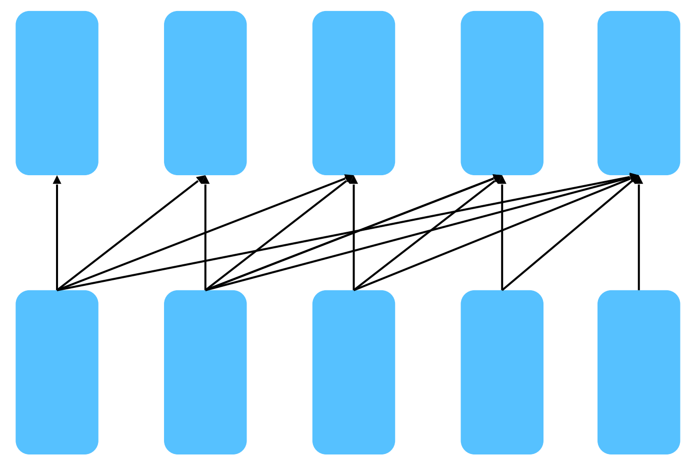
- Bidirectional; can condition on the future context

**Encoder-Decoder**



- Map two sequences of different length together

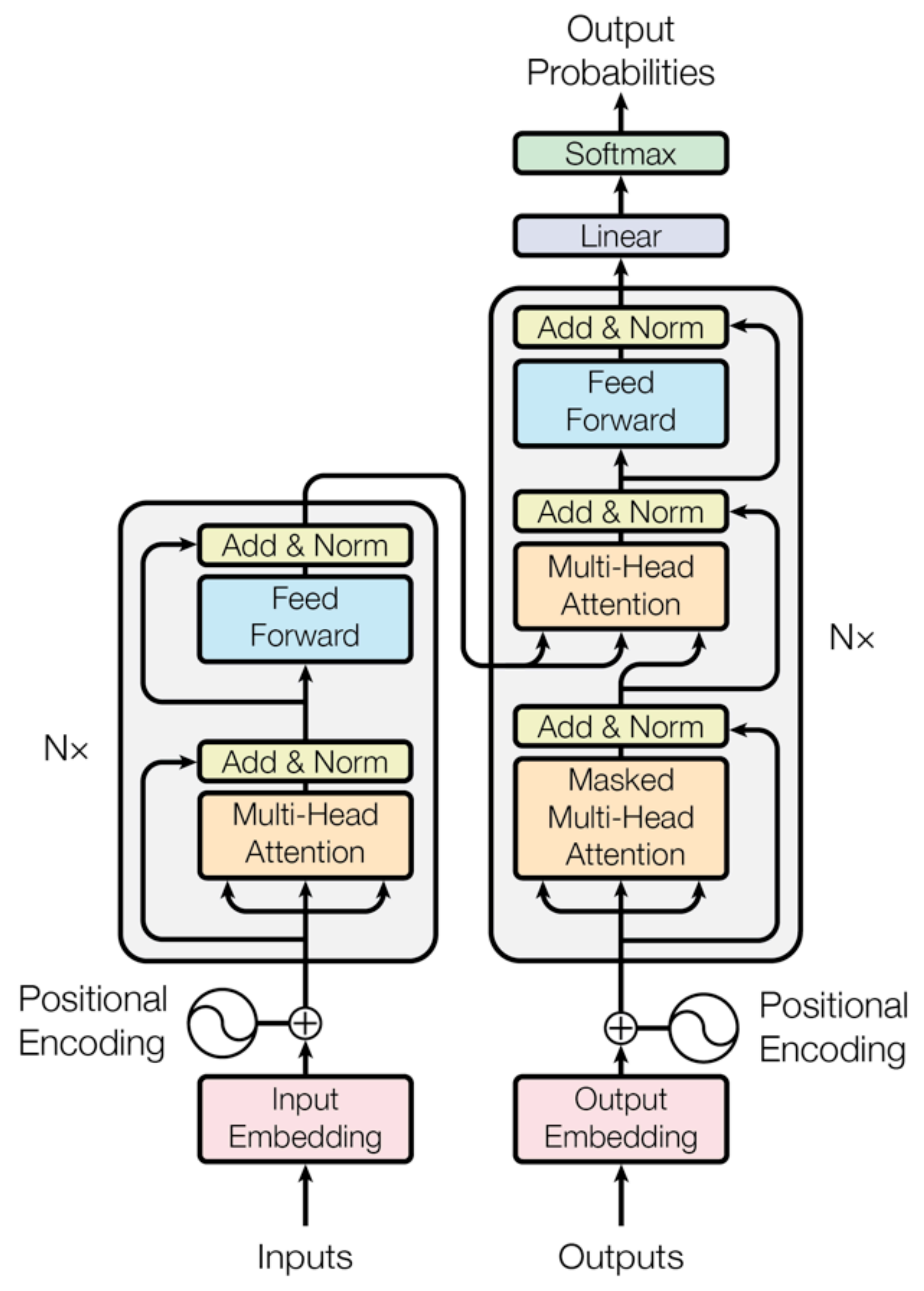
**Decoder**



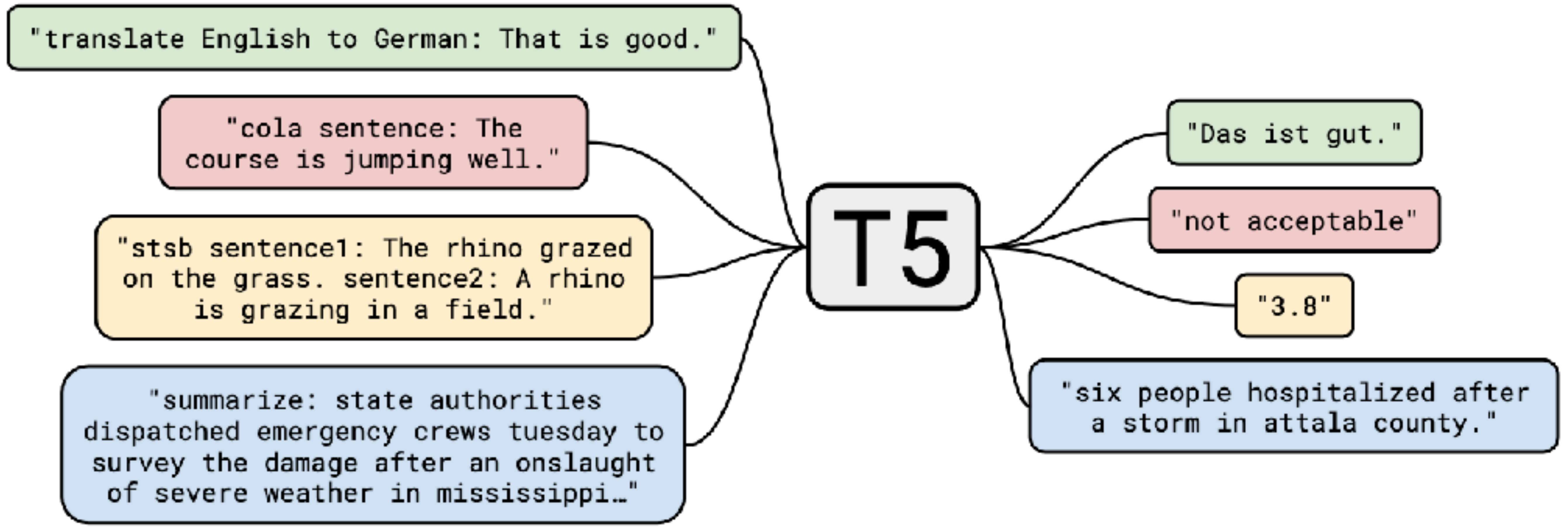
- Language modeling; can only condition on the past context

# Text-to-text models: the best of both worlds

- So bar, **encoder-only models (e.g., BERT)** enjoy the benefits of **bidirectionality** but they can't be used to generate text
- **Decoder-only models (e.g., GPT)** can do generation but they are left-to-right LMs..
- Text-to-text models combine the best of both worlds!



T5 = <u>T</u>ext-<u>t</u>o-<u>T</u>ext <u>T</u>ransfer <u>T</u>ransformer

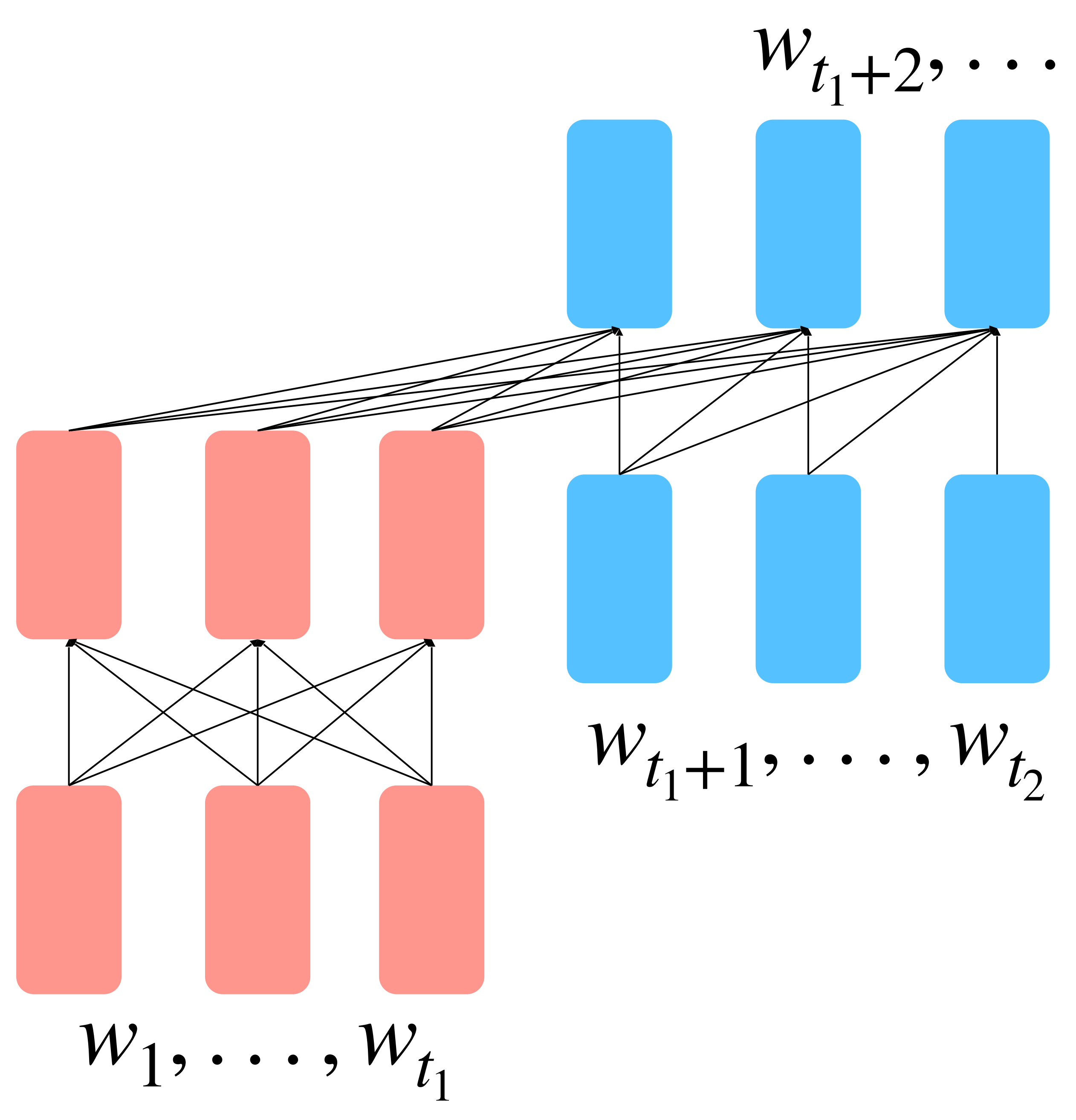


"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

(Raffel et al., 2020): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

# Encoder-decoder: architecture

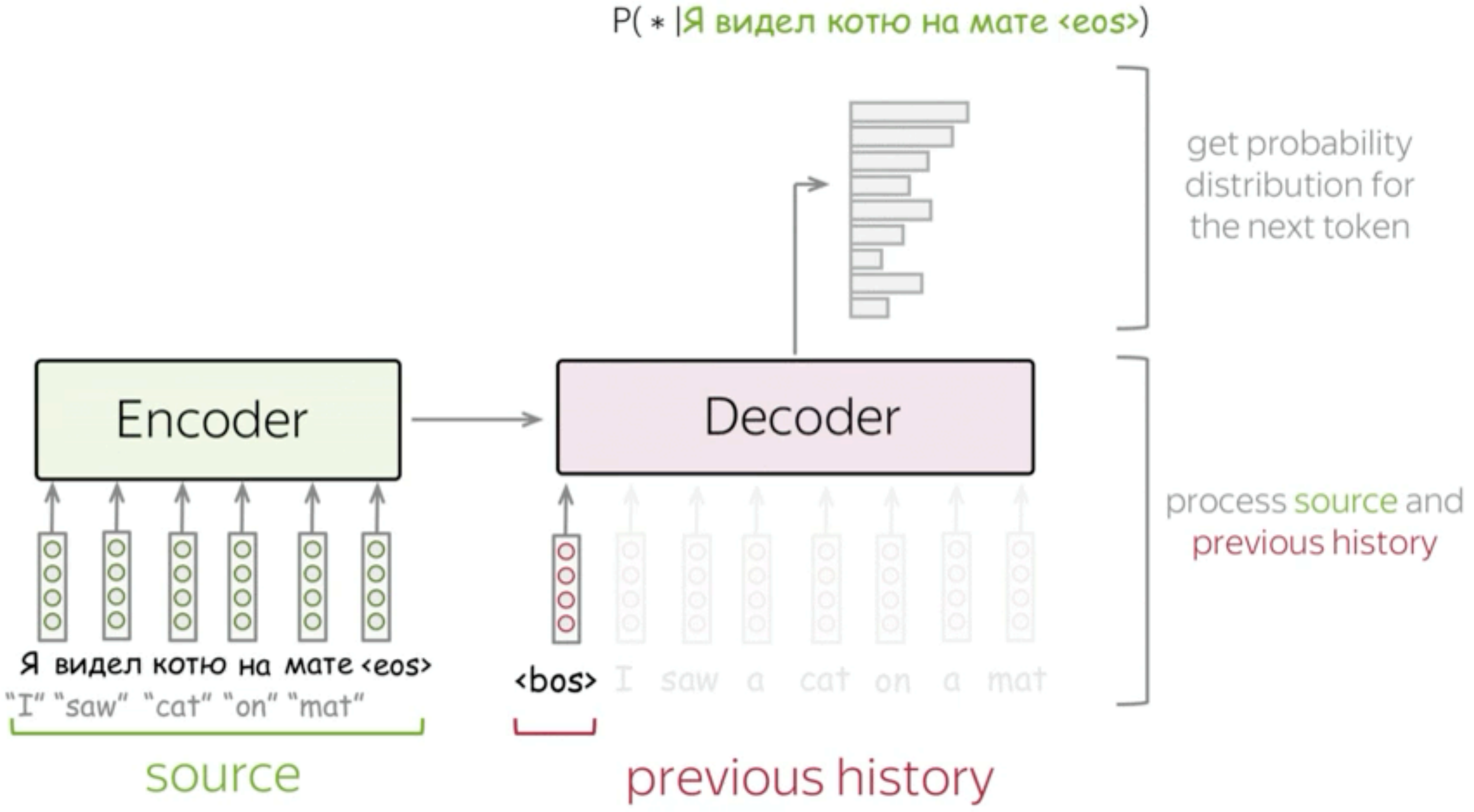
- Moving towards **open-text generation**…
- **Encoder** builds a representation of the source and gives it to the **decoder**
- **Decoder** uses the source representation to generate the target sentence
- The **encoder** portion benefits from **bidirectional** context; the **decoder** portion is used to train the whole model through **language modeling**



$$h_1, \ldots, h_{t_1} = \text{Encoder}(w_1, \ldots, w_{t_1})$$
$$h_{t_1+1}, \ldots, h_{t_2} = \text{Decoder}(w_{t_1+1}, \ldots, w_{t_2}, h_1, \ldots, h_{t_1})$$
$$y_i \sim Ah_i + b, i > t$$

[Raffel et al., 2018]

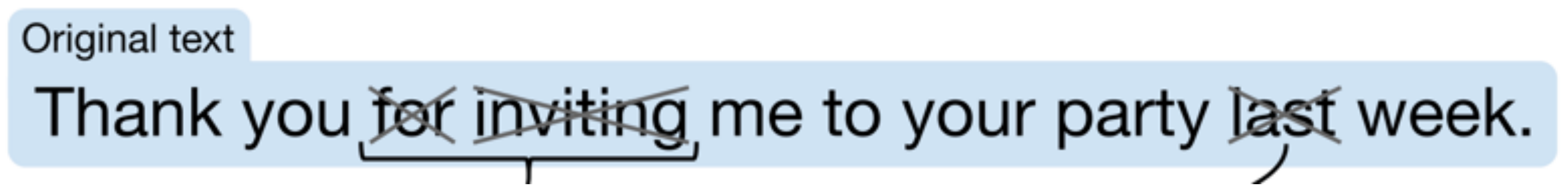
## Encoder-decoder: machine translation example

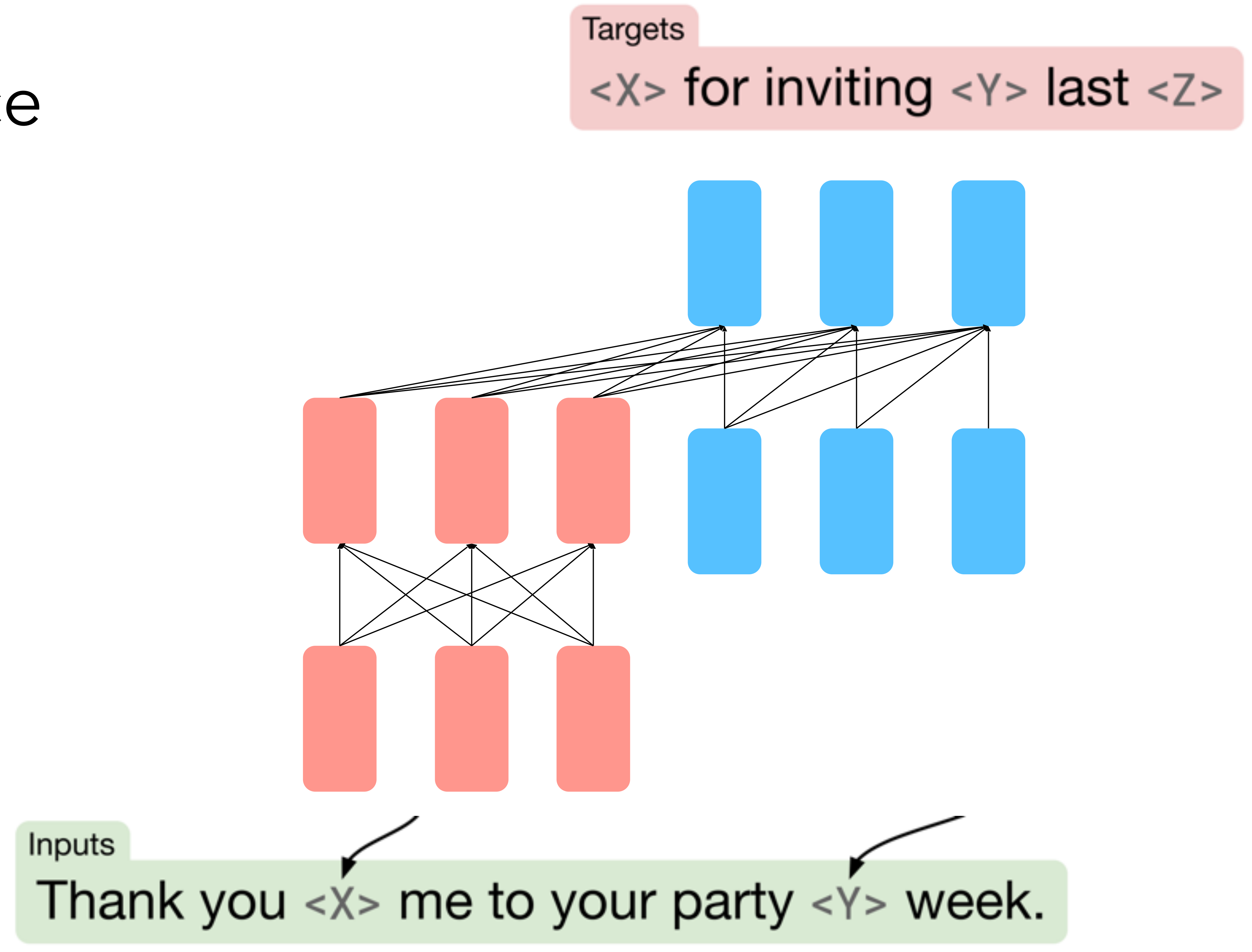

P( * |Я видел котю на мате <eos>)
get probability distribution for the next token
Encoder
Decoder
process source and previous history
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
source
<bos> I saw a cat on a mat
previous history

[Lena Viota Blog]

# Encoder-decoder: training objective

- **T5 [Raffel et al., 2018]**
- **Text span corruption (denoising):** Replace different-length spans from the input with unique placeholders (e.g., <extra_id_0>); decode out the masked spans.
    - Done during **text preprocessing**: training uses **language modeling** objective at the decoder side

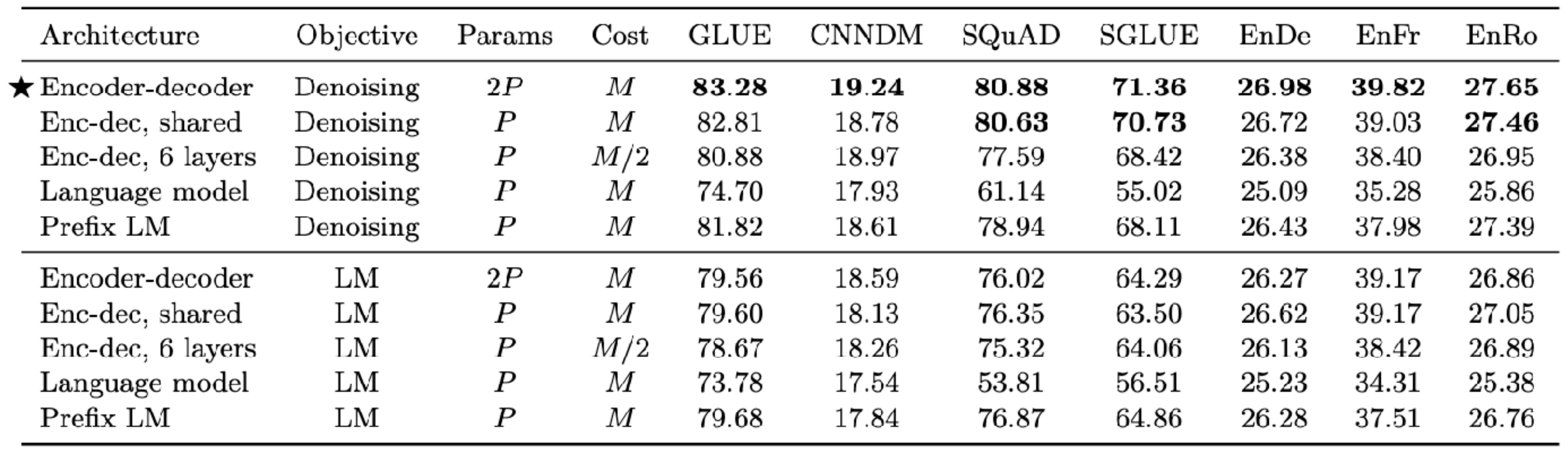



Targets
<X> for inviting <Y> last <Z>
Inputs
Thank you <X> me to your party <Y> week.

# Encoder-decoder: T5

[Raffel et al., 2018]

- **Encoder-decoders** works better than decoders
- **Span corruption (denoising)** objective works better than language modeling

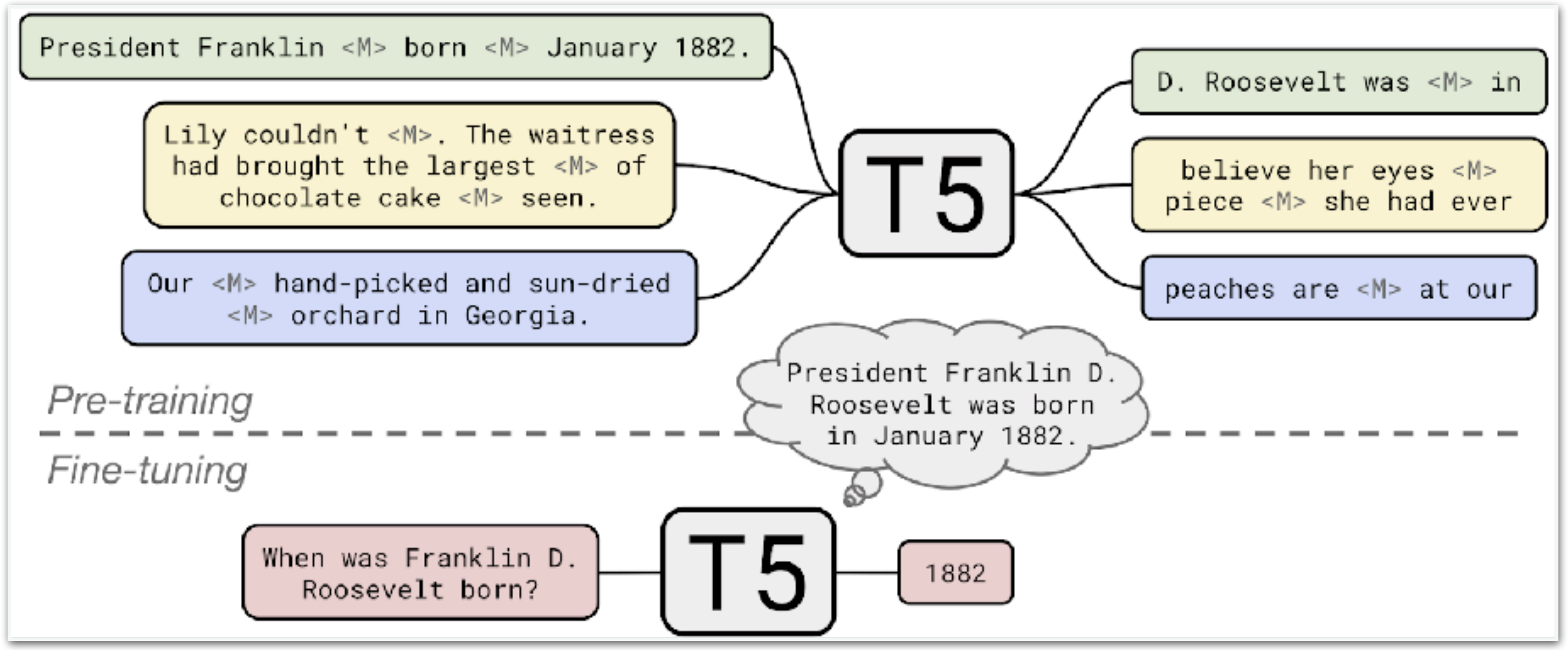


| Architecture | Objective | Params | Cost | GLUE | CNNDM | SQuAD | SGLUE | EnDe | EnFr | EnRo |
|---|---|---|---|---|---|---|---|---|---|---|
| ★ Encoder-decoder | Denoising | $2P$ | $M$ | **83.28** | **19.24** | **80.88** | **71.36** | **26.98** | **39.82** | **27.65** |
| Enc-dec, shared | Denoising | $P$ | $M$ | 82.81 | 18.78 | **80.63** | **70.73** | 26.72 | 39.03 | **27.46** |
| Enc-dec, 6 layers | Denoising | $P$ | $M/2$ | 80.88 | 18.97 | 77.59 | 68.42 | 26.38 | 38.40 | 26.95 |
| Language model | Denoising | $P$ | $M$ | 74.70 | 17.93 | 61.14 | 55.02 | 25.09 | 35.28 | 25.86 |
| Prefix LM | Denoising | $P$ | $M$ | 81.82 | 18.61 | 78.94 | 68.11 | 26.43 | 37.98 | 27.39 |
| Encoder-decoder | LM | $2P$ | $M$ | 79.56 | 18.59 | 76.02 | 64.29 | 26.27 | 39.17 | 26.86 |
| Enc-dec, shared | LM | $P$ | $M$ | 79.60 | 18.13 | 76.35 | 63.50 | 26.62 | 39.17 | 27.05 |
| Enc-dec, 6 layers | LM | $P$ | $M/2$ | 78.67 | 18.26 | 75.32 | 64.06 | 26.13 | 38.42 | 26.89 |
| Language model | LM | $P$ | $M$ | 73.78 | 17.54 | 53.81 | 56.51 | 25.23 | 34.31 | 25.38 |
| Prefix LM | LM | $P$ | $M$ | 79.68 | 17.84 | 76.87 | 64.86 | 26.28 | 37.51 | 26.76 |

## Encoder-decoder: T5

[Raffel et al., 2018]

- **Text-to-Text:** convert NLP tasks into input/output text sequences
- **Dataset:** Colossal Clean Crawled Corpus (C4), 750G text data!
- **Various Sized Models:**
  - Base (222M)
  - Small (60M)
  - Large (770M)
  - 3B
  - 11B
- **Achieved SOTA with scaling & purity of data**



President Franklin <M> born <M> January 1882.
Lily couldn't <M>. The waitress had brought the largest <M> of chocolate cake <M> seen.
Our <M> hand-picked and sun-dried <M> orchard in Georgia.
T5
D. Roosevelt was <M> in
believe her eyes <M> piece <M> she had ever
peaches are <M> at our
Pre-training
Fine-tuning
President Franklin D. Roosevelt was born in January 1882.
When was Franklin D. Roosevelt born?
T5
1882

[Google Blog]

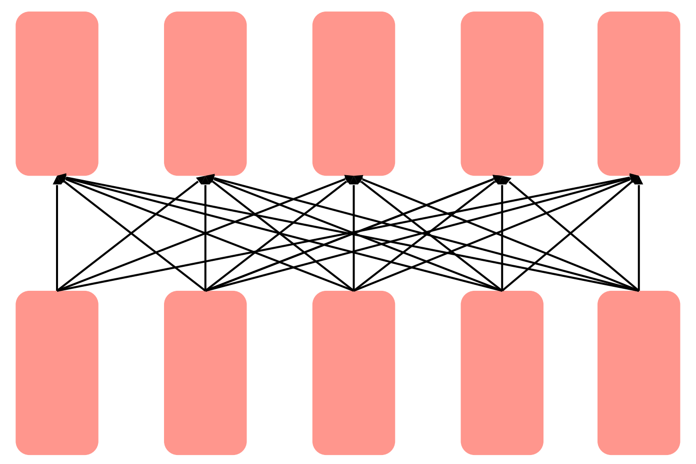
## Encoder-decoder: pros & cons

- A nice middle ground between leveraging **bidirectional** contexts and **open-text** generation
- Good for **multi-task** fine-tuning

- Require more **text wrangling**
- **Harder to train**
- **Less flexible** for natural language generation

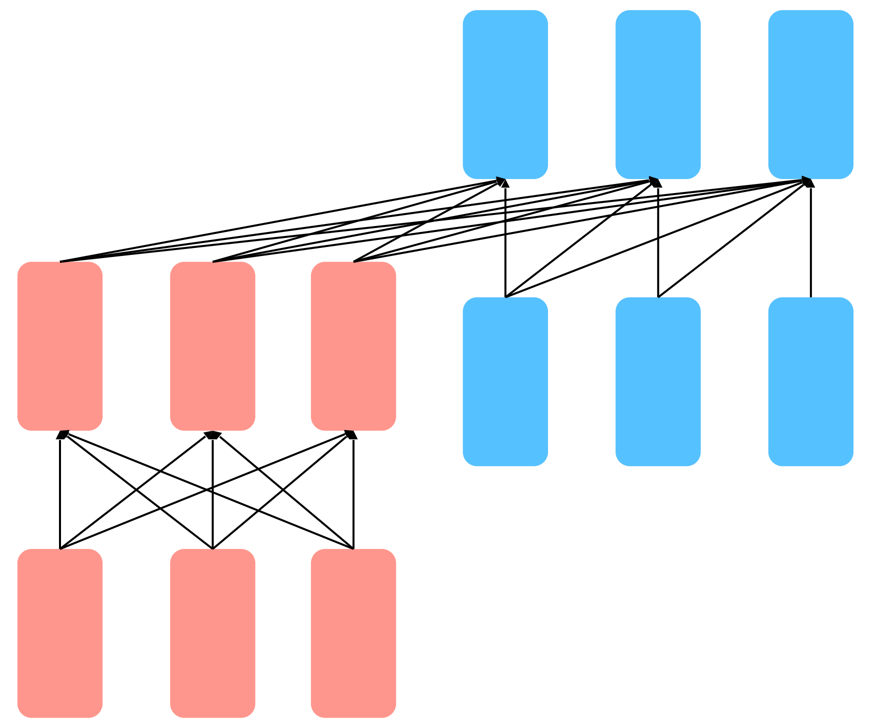
# Pre-training architectures

**Encoder**



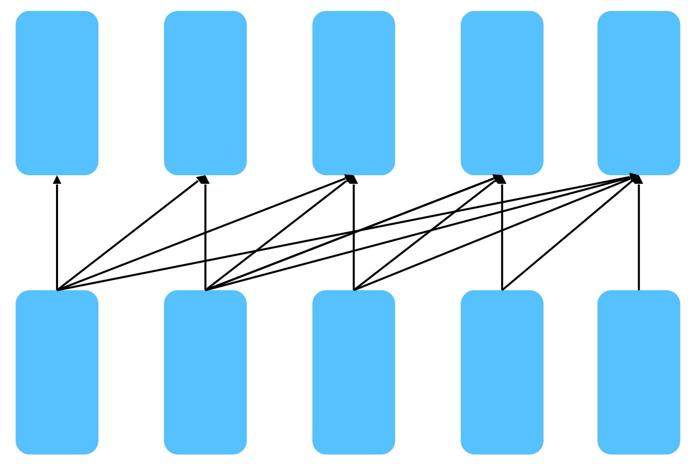
- Bidirectional; can condition on the future context

**Encoder-Decoder**



- Map two sequences of different length together

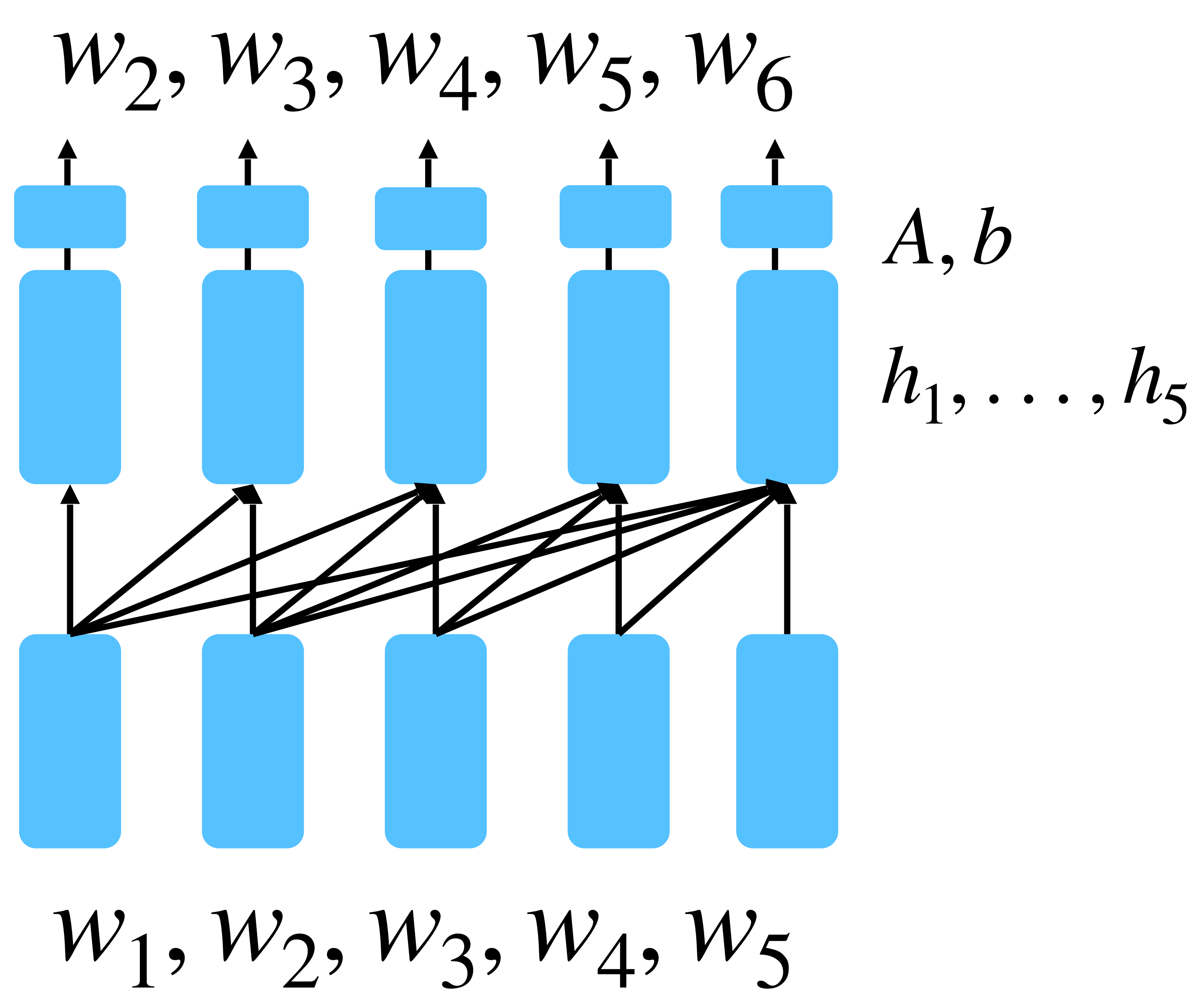
**Decoder**



- Language modeling; can only condition on the past context

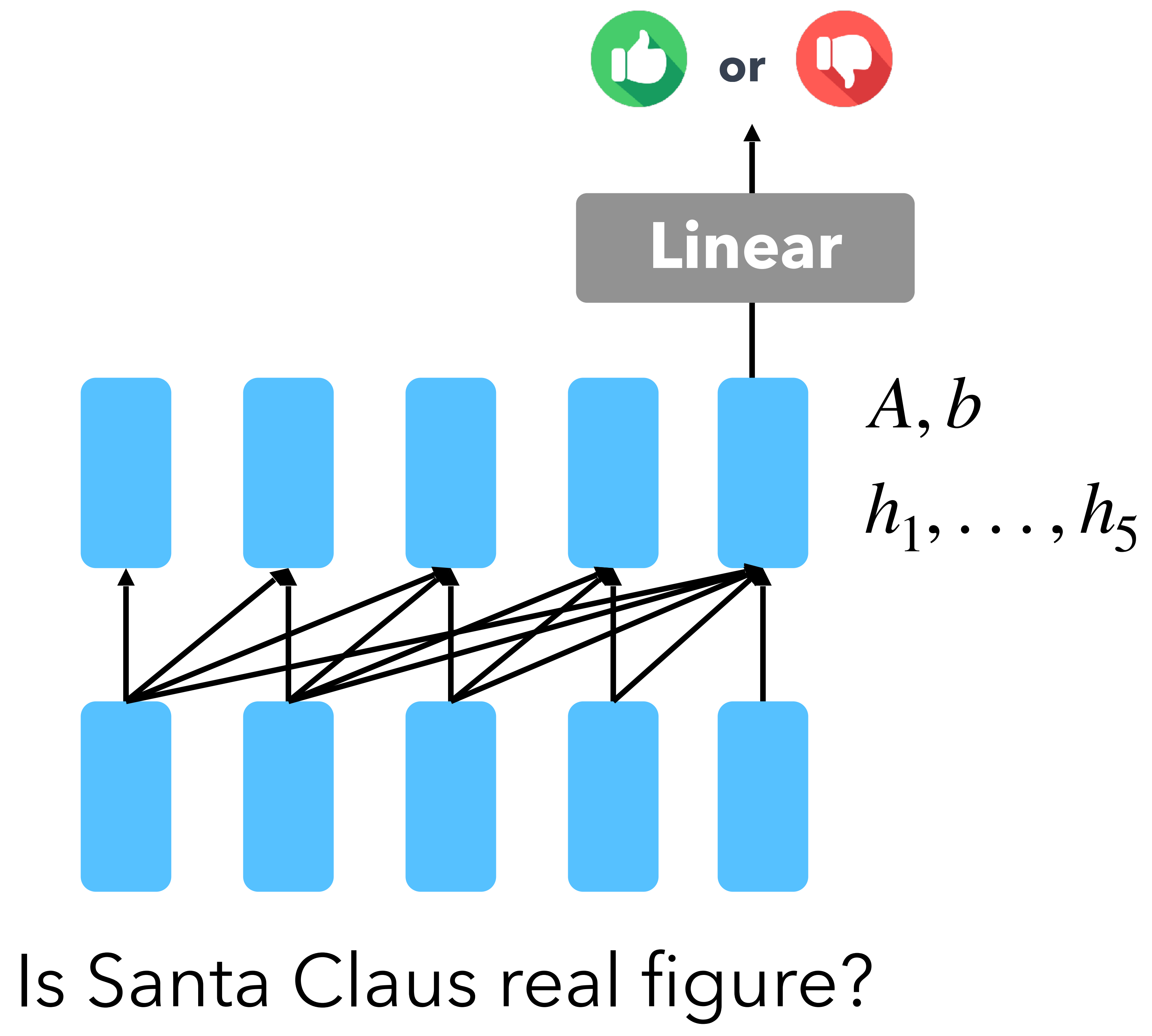
# Decoder: training objective

- Many most famous generative LLMs are **decoder-only**
  - e.g., GPT1/2/3/4, Llama1/2
- **Language modeling!** Natural to be used for **open-text generation**
- **Conditional LM:** $p(w_t \mid w_1, \ldots, w_{t-1}, x)$
  - Conditioned on a source context $x$ to generate from left-to-right
- Can be fine-tuned for **natural language generation (NLG)** tasks, e.g., dialogue, summarization.



# Decoder: training objective

- Customizing the pre-trained model for downstream tasks:
  - Add a **linear layer** on top of the last hidden layer to make it a classifier!
  - During fine-tuning, trained the randomly **initialized linear layer**, along with **all parameters** in the neural net.



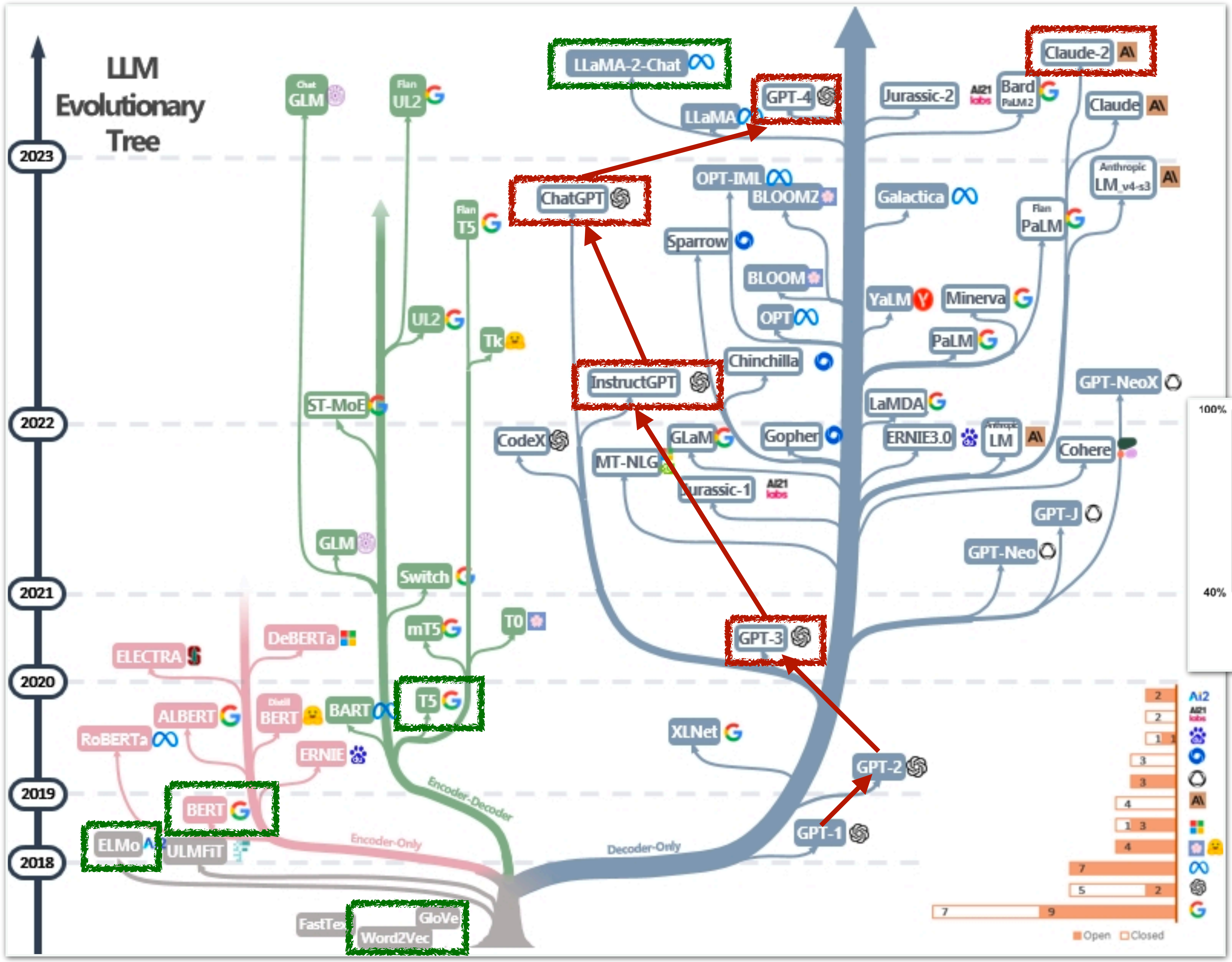
## Neural language models: scaling laws
## From GPT1 to GPT4

# Evolution tree of pretrained LMs

~200 billion

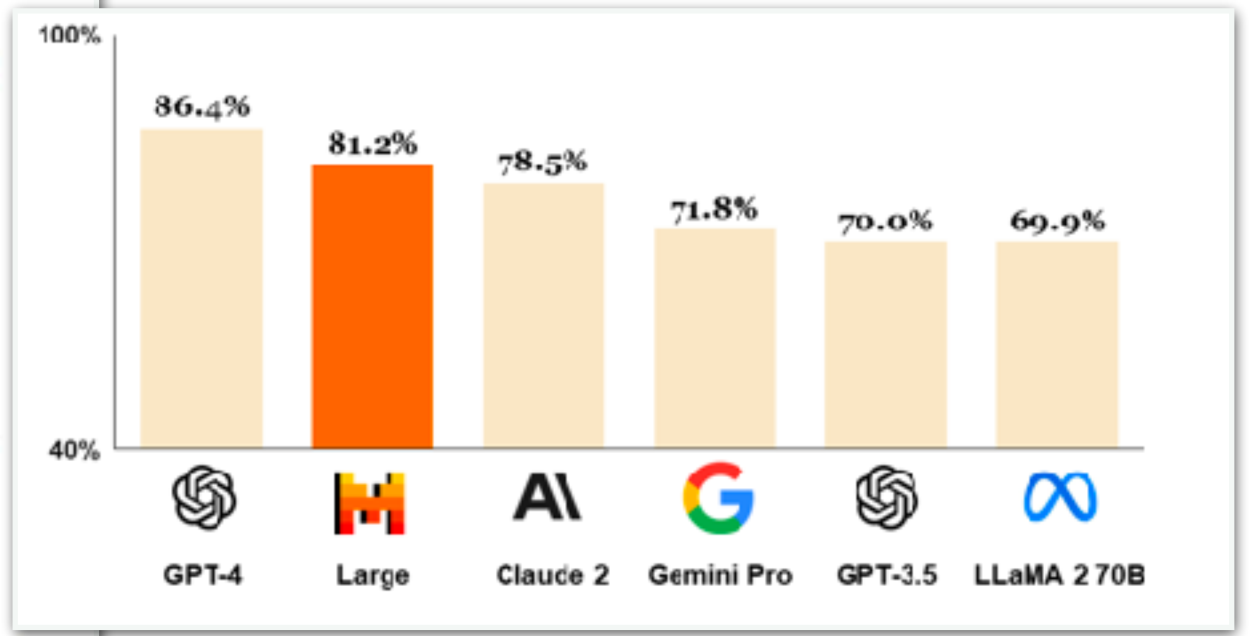
Model size
(# of parameters)
~1000 times larger

~300 million



🟩 Open-sourced

🟥 Close-sourced



https://github.com/Mooler0410/LLMsPracticalGuide
https://mistral.ai/news/mistral-large/

## From GPT1 to GPT-2 to GPT-3

- All **decoder-only Transformer-based language models**
- Model size ↑, training corpora ↑

GPT-2

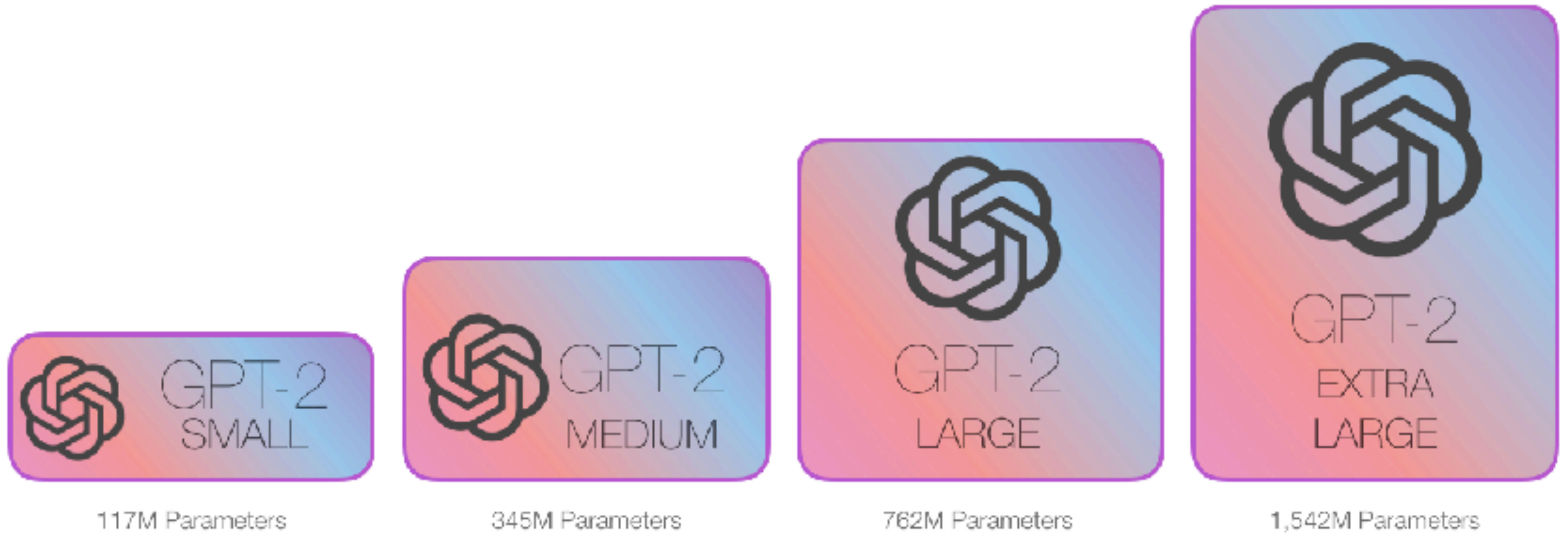
Better language models and their implications

February 14, 2019

Read paper ↗

View code ↗

Context size = 1024

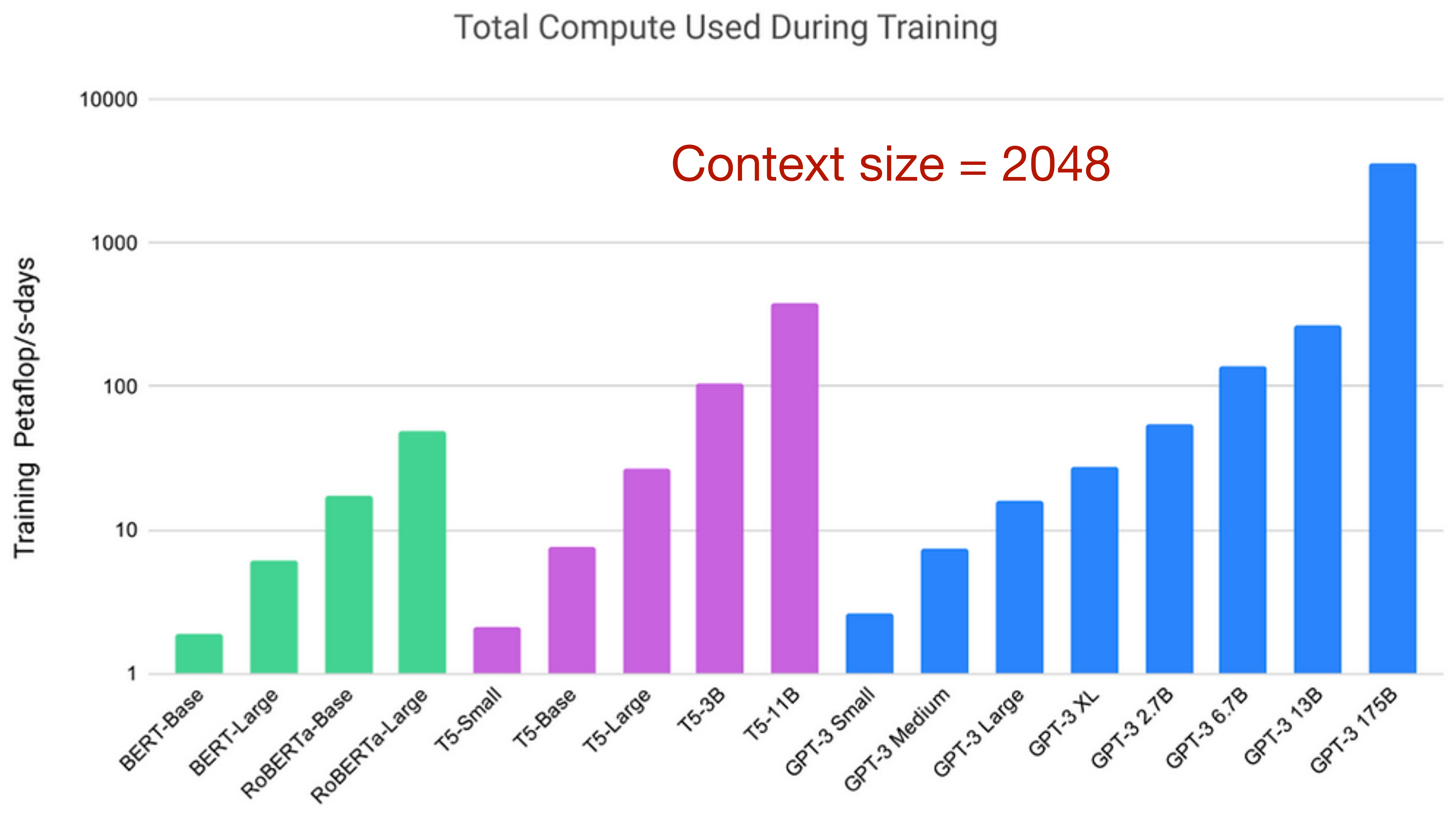


.. trained on 40Gb of Internet text ..

(Radford et al., 2019): Language Models are Unsupervised Multitask Learners

# GPT-3: language models are few-shot learners

- GPT-2 → GPT-3: 1.5B → 175B (# of parameters), ~14B → 300B (# of tokens)



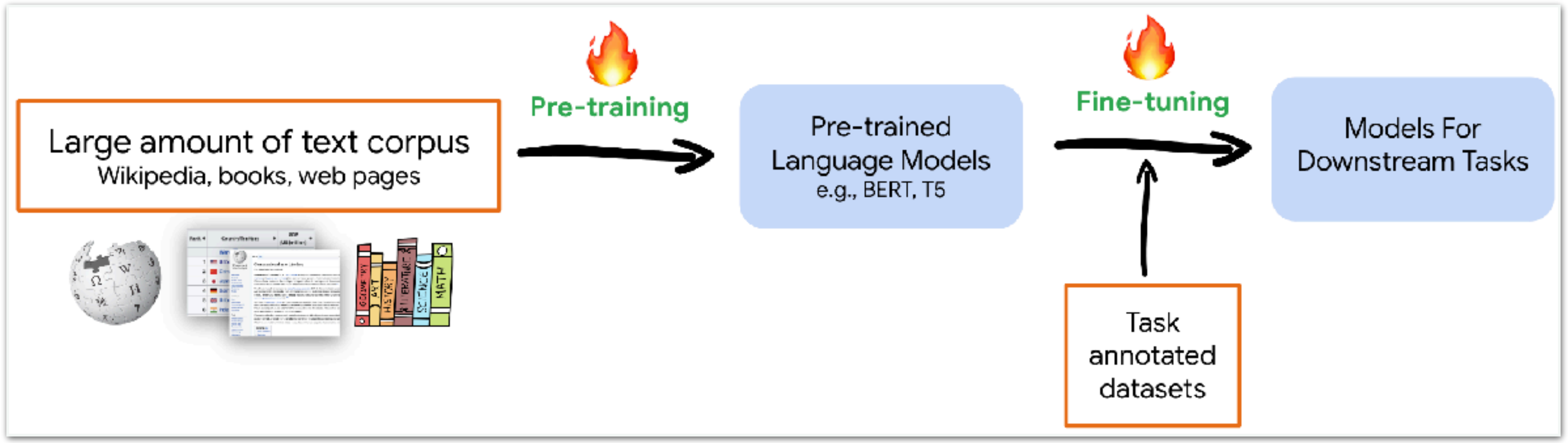
(Brown et al., 2020): Language Models are Few-Shot Learners

Training computation is measured using floating-point operations or "FLOP".

One FLOP represents a single arithmetic operation involving floating-point numbers, such as addition, subtraction, multiplication, or division.

# Before GPT3: Modern learning paradigm

- **Pre-training + supervised training/fine-tuning**
  - First train Transformer using a lot of general text using unsupervised learning. This is called **pretraining**.
  - Then train the pretrained Transformer for a specific task using supervised learning. This is called **finetuning**.

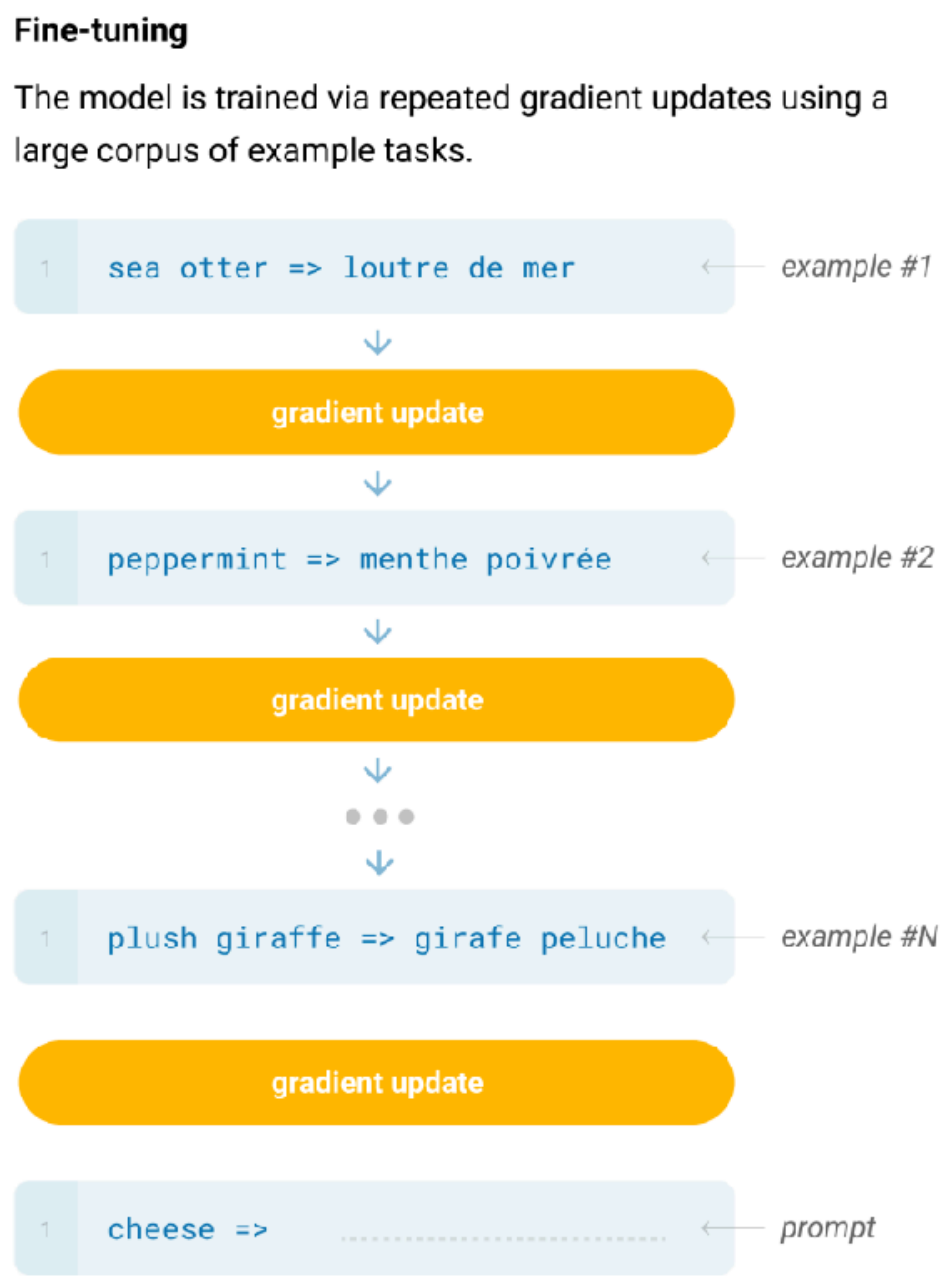

Large amount of text corpus
Wikipedia, books, web pages
Pre-training
Pre-trained Language Models
e.g., BERT, T5
Fine-tuning
Models For Downstream Tasks
Task annotated datasets

# Paradigm shift since GPT-3

- Before GPT-3, **Pre-training + supervised training/fine-tuning** is the default way of doing learning in models like BERT/T5/GPT-2
  - SST-2 has 67k examples, SQuAD has 88k (passage, answer, question) triples

- Fine-tuning requires computing the gradient and applying a parameter update on every example (or every K examples in a mini-batch)

- However, this is very expensive for the 175B GPT-3 model


**Fine-tuning**

The model is trained via repeated gradient updates using a large corpus of example tasks.

```
sea otter => loutre de mer
```

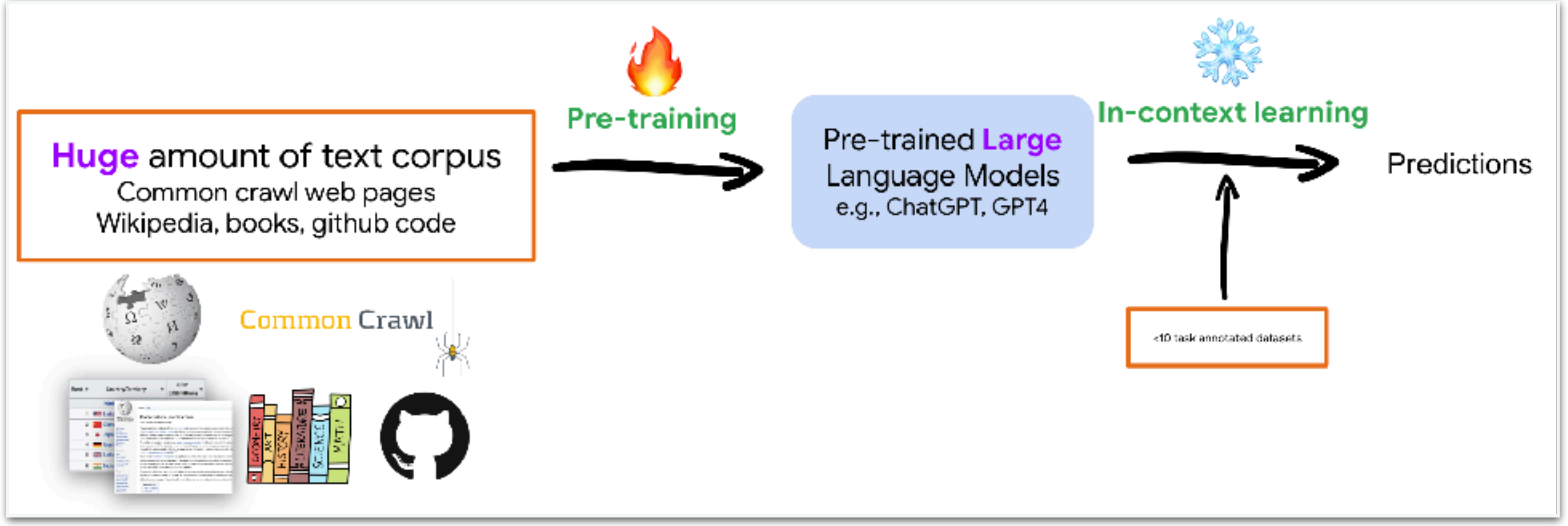
```
peppermint => menthe poivrée
```

```
plush giraffe => girafe peluche
```

```
cheese =>
```

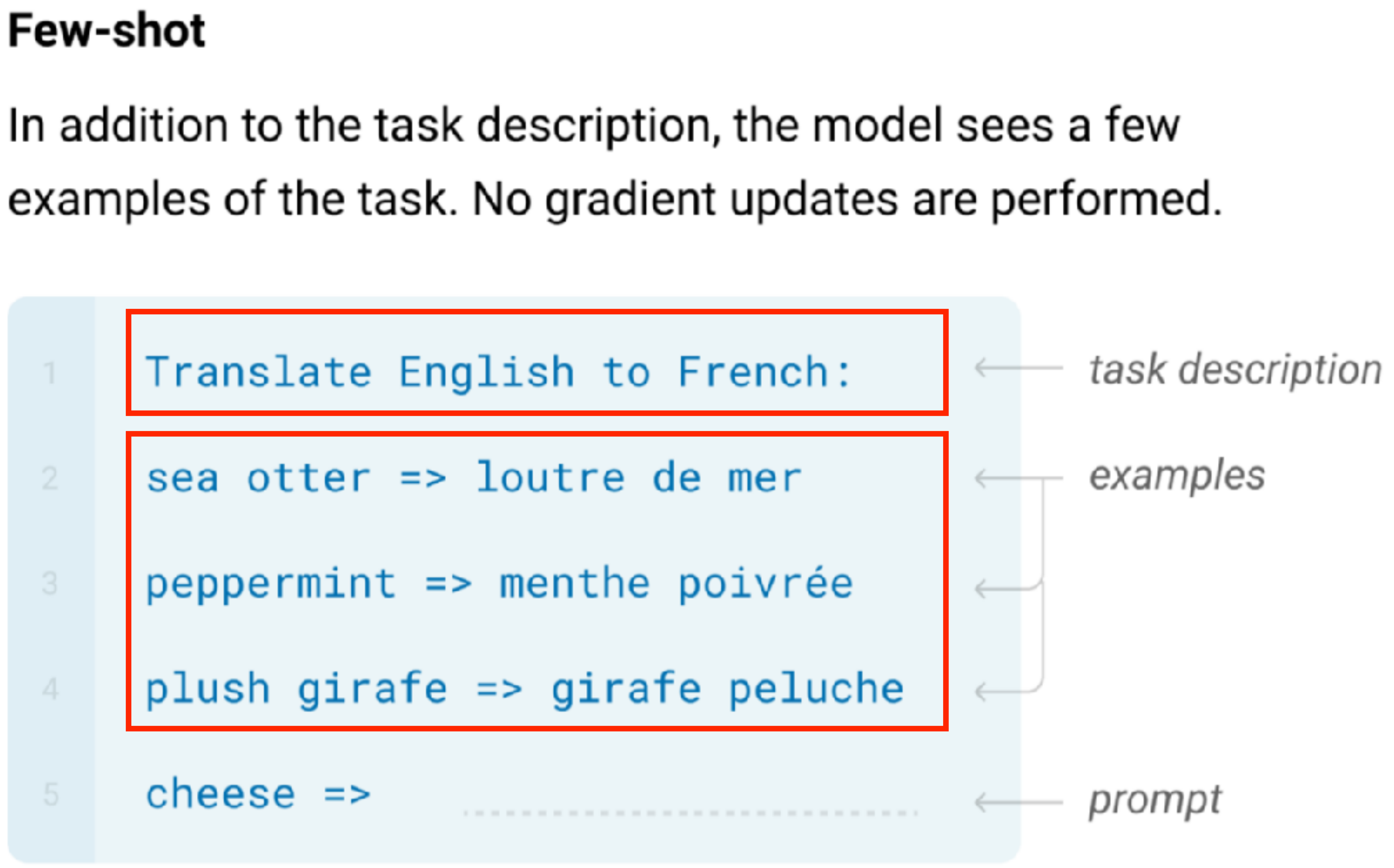
# Latest learning paradigm shift since GPT-3

- **Pre-training + prompting/in-context learning (no training this step)**
  - First train a **large (>7~175B)** Transformer using a lot of general text using unsupervised learning. This is called **large** language model **pretraining**.
  - Then **directly use** the pretrained large Transformer (**no further finetuning/training**) for any different task given only a natural language description of the task or a few task (x, y) examples. This is called **prompting/in-context learning**.


Huge amount of text corpus
Common crawl web pages
Wikipedia, books, github code
Common Crawl
Pre-training
Pre-trained Large Language Models
e.g., ChatGPT, GPT4
In-context learning
Predictions
<10 task annotated datasets

## GPT-3: few-shot in-context learning

- GPT-3 proposes an alternative: **in-context learning**



**Few-shot**

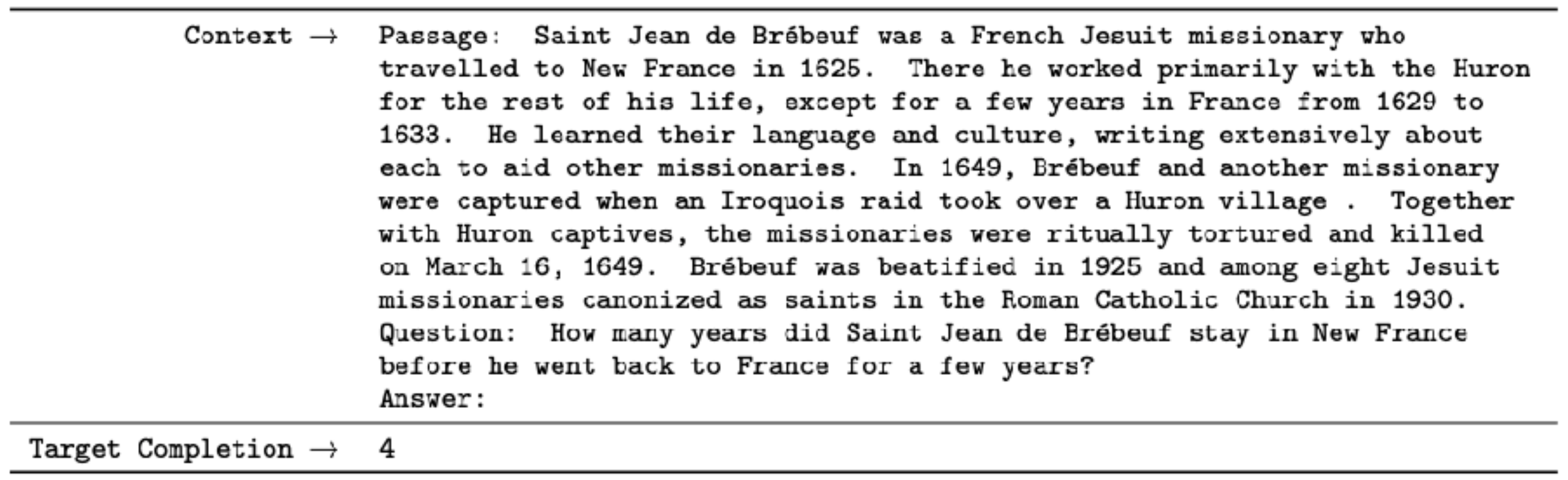
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

- This is just a forward pass, **no gradient update at all**!
- You only need to feed a small number of examples (e.g., 32)

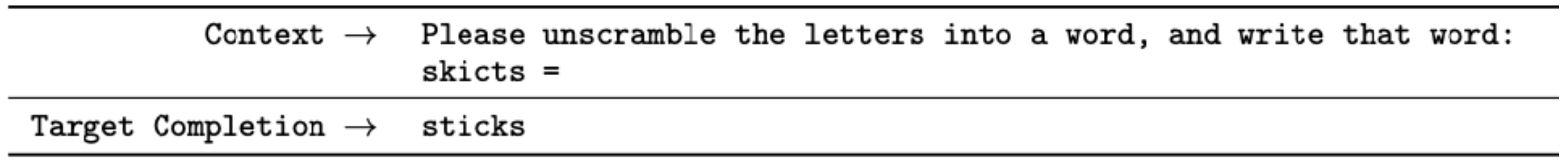
(On the other hand, you can’t feed many examples at once too as it is bounded by context size)

# GPT-3: task specifications



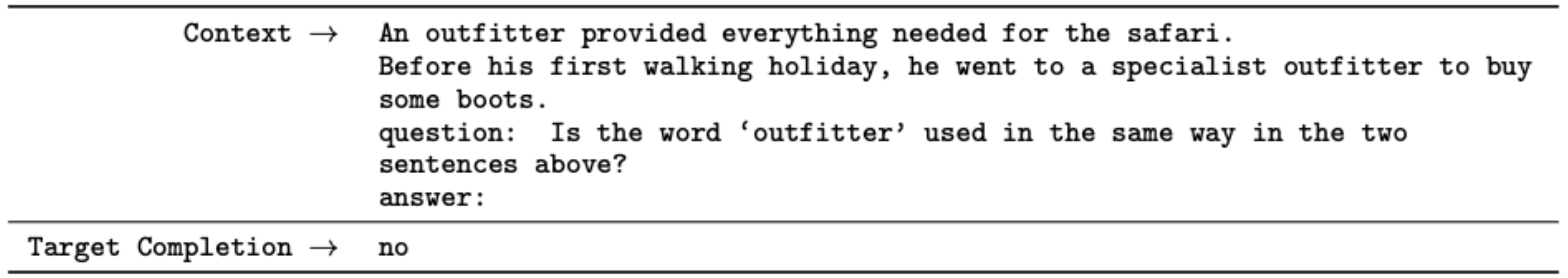
| | |
|---|---|
| Context → | Passage: Saint Jean de Brébeuf was a French Jesuit missionary who travelled to New France in 1625. There he worked primarily with the Huron for the rest of his life, except for a few years in France from 1629 to 1633. He learned their language and culture, writing extensively about each to aid other missionaries. In 1649, Brébeuf and another missionary were captured when an Iroquois raid took over a Huron village . Together with Huron captives, the missionaries were ritually tortured and killed on March 16, 1649. Brébeuf was beatified in 1925 and among eight Jesuit missionaries canonized as saints in the Roman Catholic Church in 1930.<br>Question: How many years did Saint Jean de Brébeuf stay in New France before he went back to France for a few years?<br>Answer: |
| Target Completion → | 4 |

DROP
(a reading comprehension task)



| | |
|---|---|
| Context → | Please unscramble the letters into a word, and write that word:<br>skicts = |
| Target Completion → | sticks |

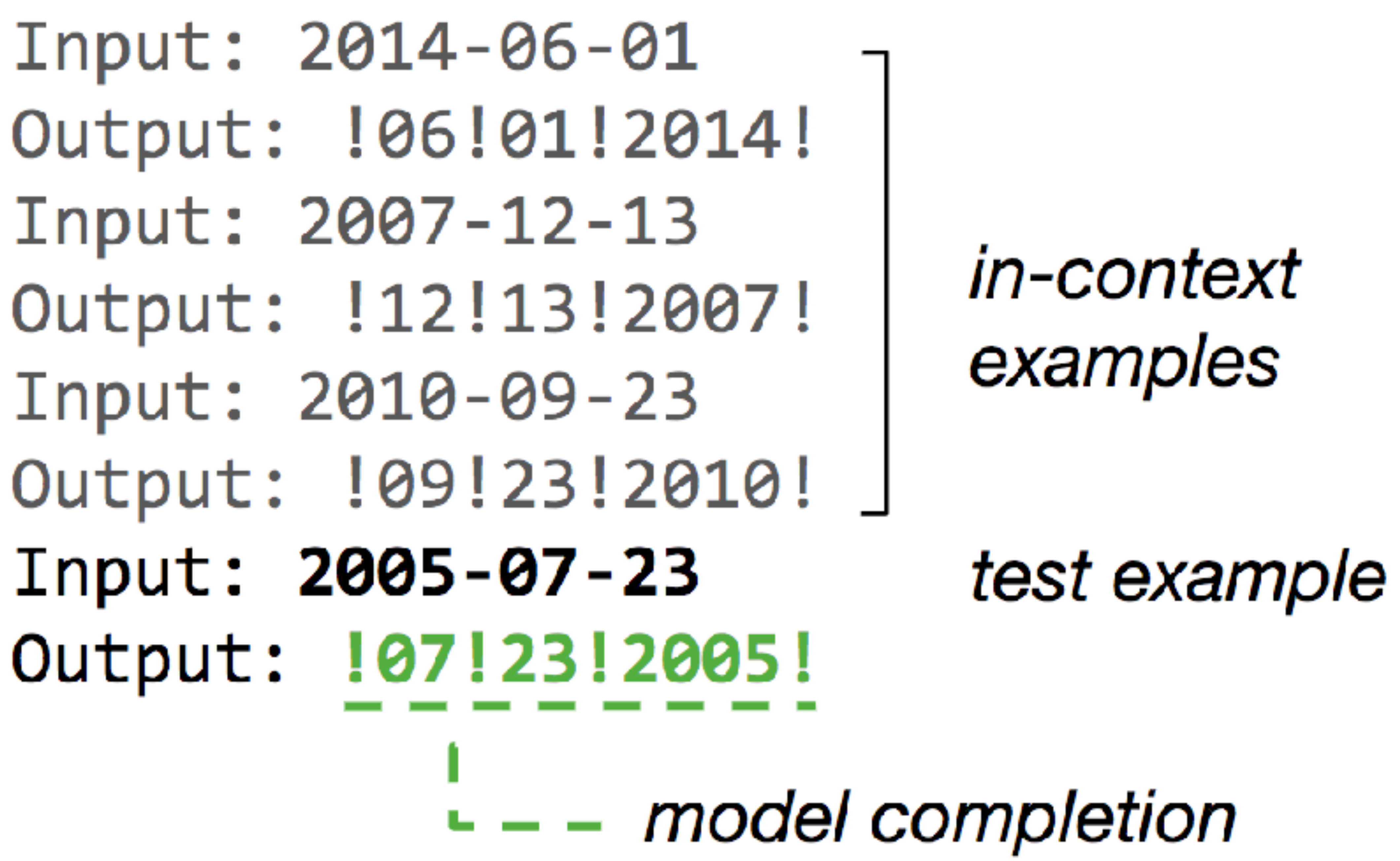
Unscrambling words



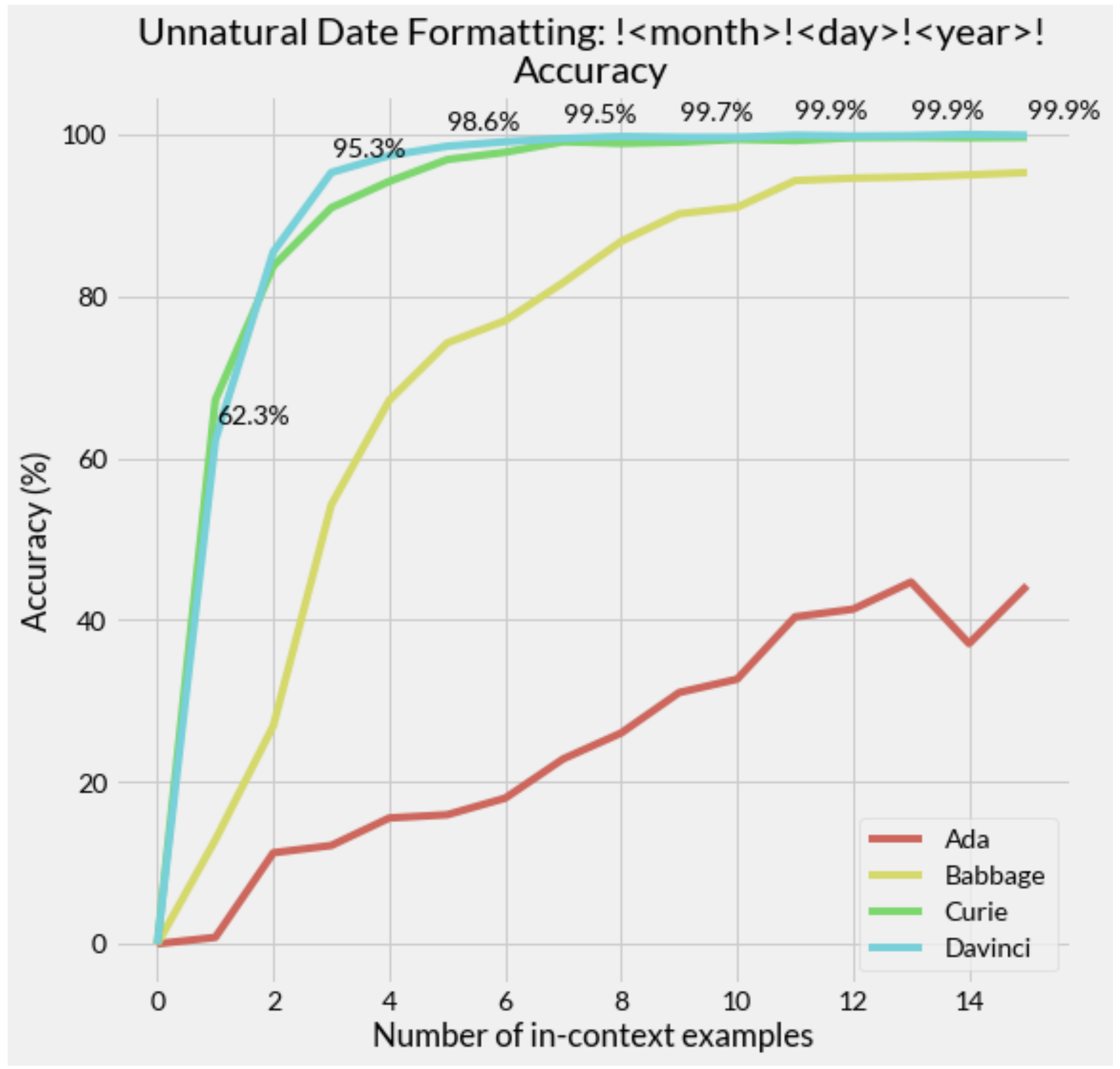
| | |
|---|---|
| Context → | An outfitter provided everything needed for the safari.<br>Before his first walking holiday, he went to a specialist outfitter to buy some boots.<br>question: Is the word ‘outfitter’ used in the same way in the two sentences above?<br>answer: |
| Target Completion → | no |

Word in context (WiC)

# GPT-3's in-context learning

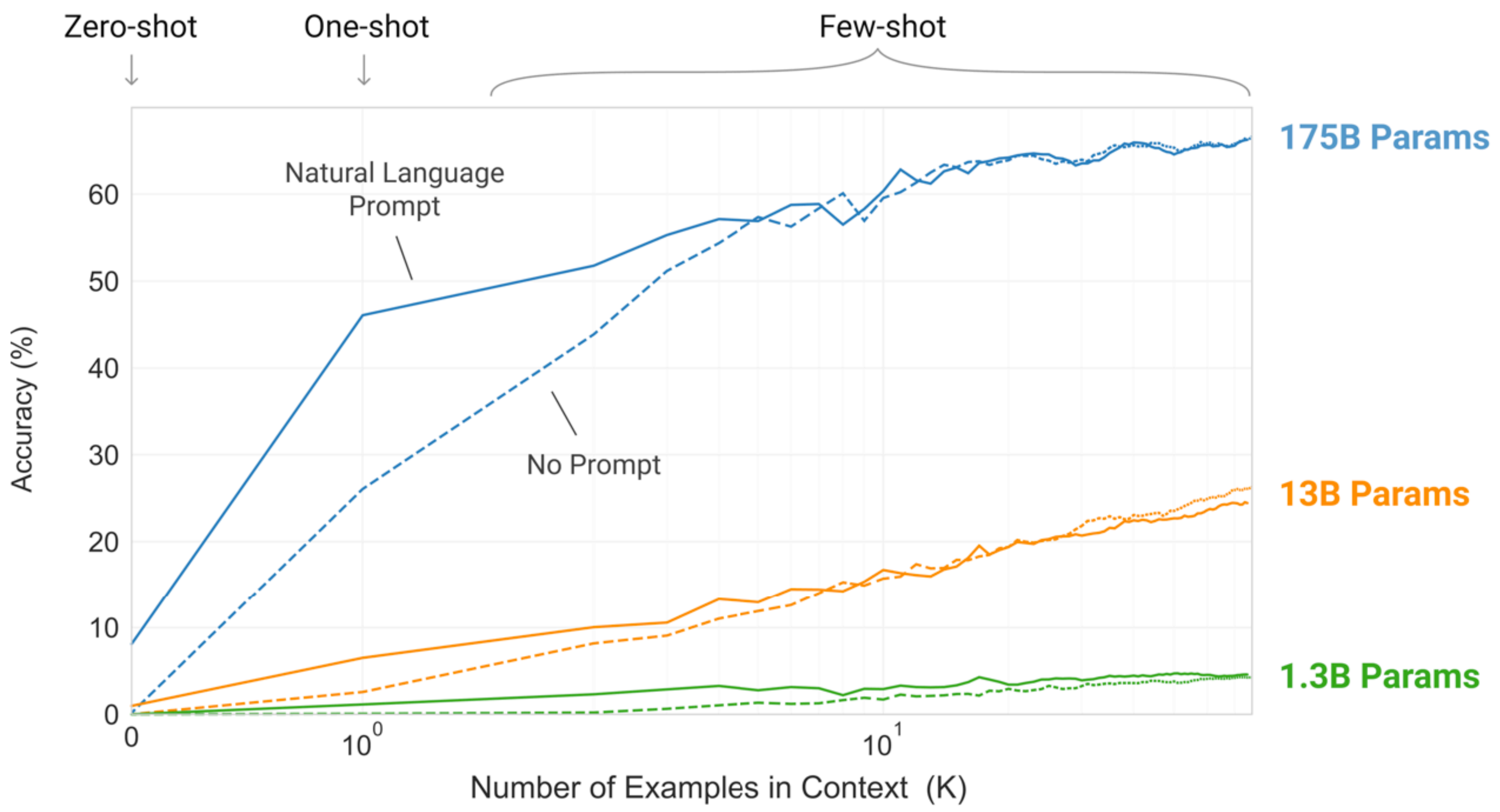


```
Input:  2014-06-01
Output:  !06!01!2014!
Input:  2007-12-13
Output:  !12!13!2007!
Input:  2010-09-23
Output:  !09!23!2010!
Input:  2005-07-23
Output:  !07!23!2005!
```

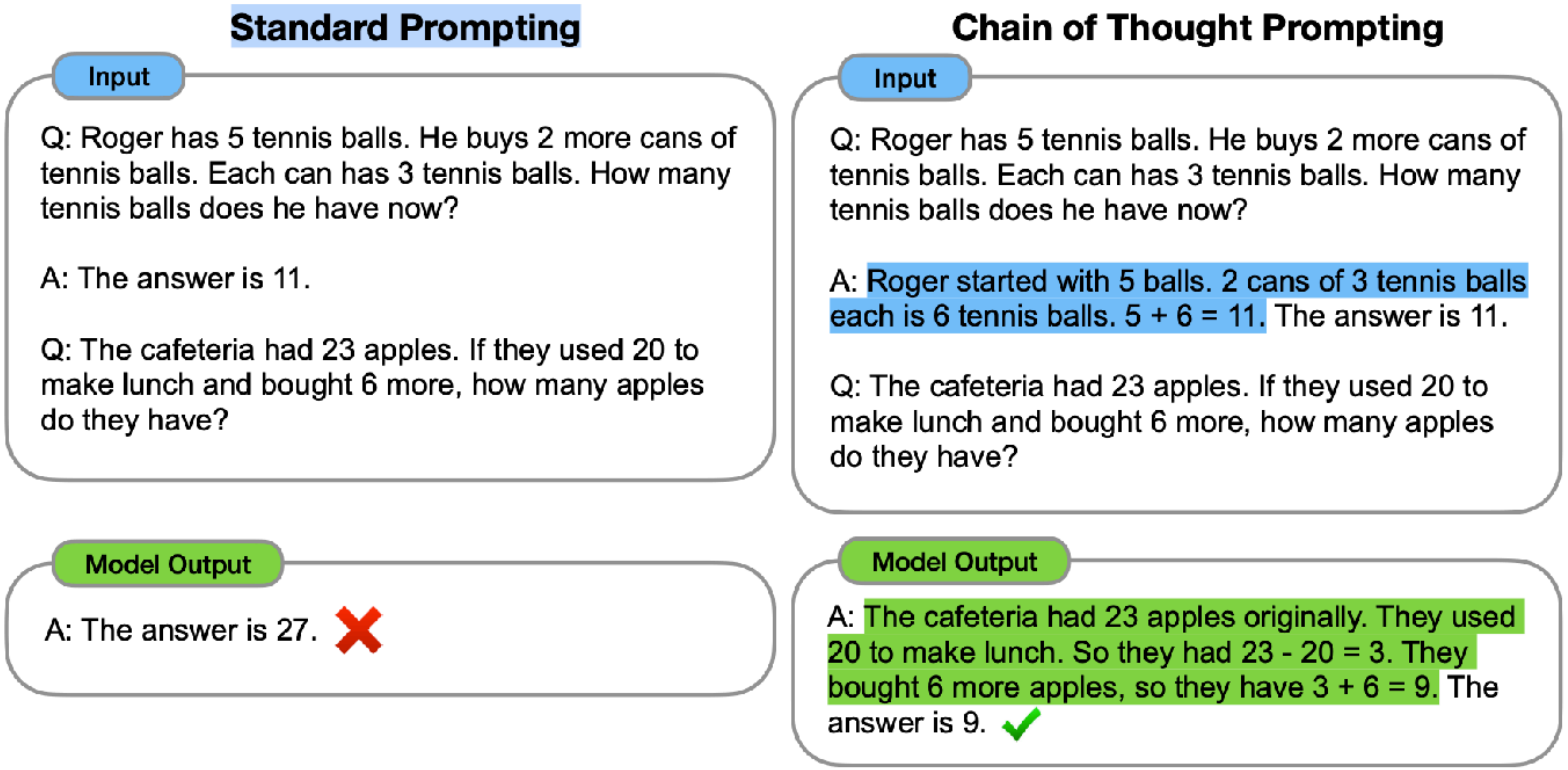


http://ai.stanford.edu/blog/in-context-learning/

# GPT-3's scaling laws in performance



# Chain-of-thought (CoT) prompting



**Standard Prompting**

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

**Chain of Thought Prompting**

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

(Wei et al., 2022): Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

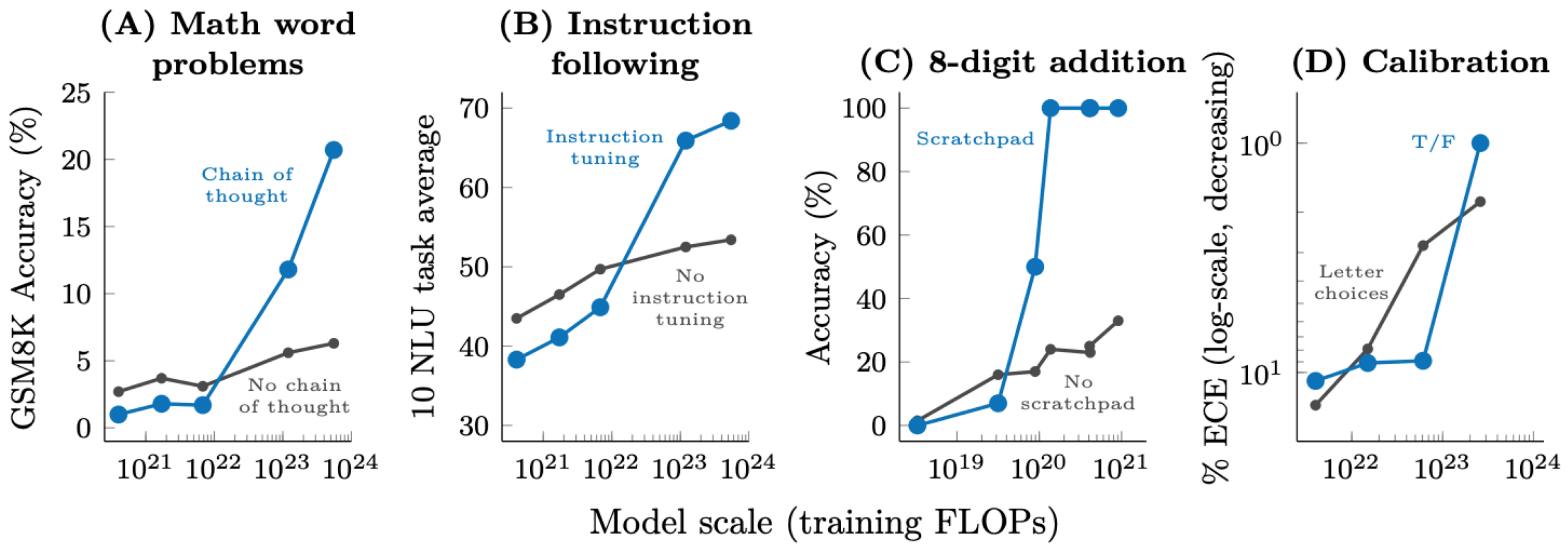
## Why in-context learning with LLMs?

- Amazing zero/few-shot performance
  - Save a lot of annotation! 🎉
- Easy to use without training
  - Just talk to them! 👍
- One model for many NLP applications 😄
  - No need to annotate and fine-tune for different tasks

But, again, they are sensitive to prompts! Need to design a good prompt or train a good example retriever! 😂

## Emergent capabilities of LLMs?

# Emergent properties of LLMs



(Wei et al., 2022) Emergent Abilities of Large Language Models

# Emergent capabilities a mirage?

- (Schaeffer et al., 2023) take issue with the characterization of "emergent capabilities"
- Most metrics used in (Wei et al., 2022) were "hard" metrics which don't give partial credit like accuracy

**Are Emergent Abilities of Large Language Models a Mirage?**

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo

Computer Science, Stanford University

Hard Accuracy:

A) 123 + 456 = 579 ✅

B) 123 + 456 = 578 ❌

C) 123 + 456 = 42 ❌

In (Wei et al., 2022), B and C are both wrong, even though B is much closer to correct than C

# Emergent capabilities a mirage?

- (Schaeffer et al., 2023) measure soft metrics (e.g., how many digits are correct, probability of the right answer) for "emergent abilities"
- Find much more predictable scaling
- Different metric choices lead to different appearances of "emergent" or not emergent
- "Emergent abilities" are a mirage(?)

Hard Accuracy:

A) 123 + 456 = 579 ✅

B) 123 + 456 = 578 ❌

C) 123 + 456 = 42 ❌

Soft Accuracy (# correct digits):

A) 123 + 456 = 579 3/3 ✅

B) 123 + 456 = 578 2/3 🤷‍♀️

C) 123 + 456 = 42 0/3 ❌