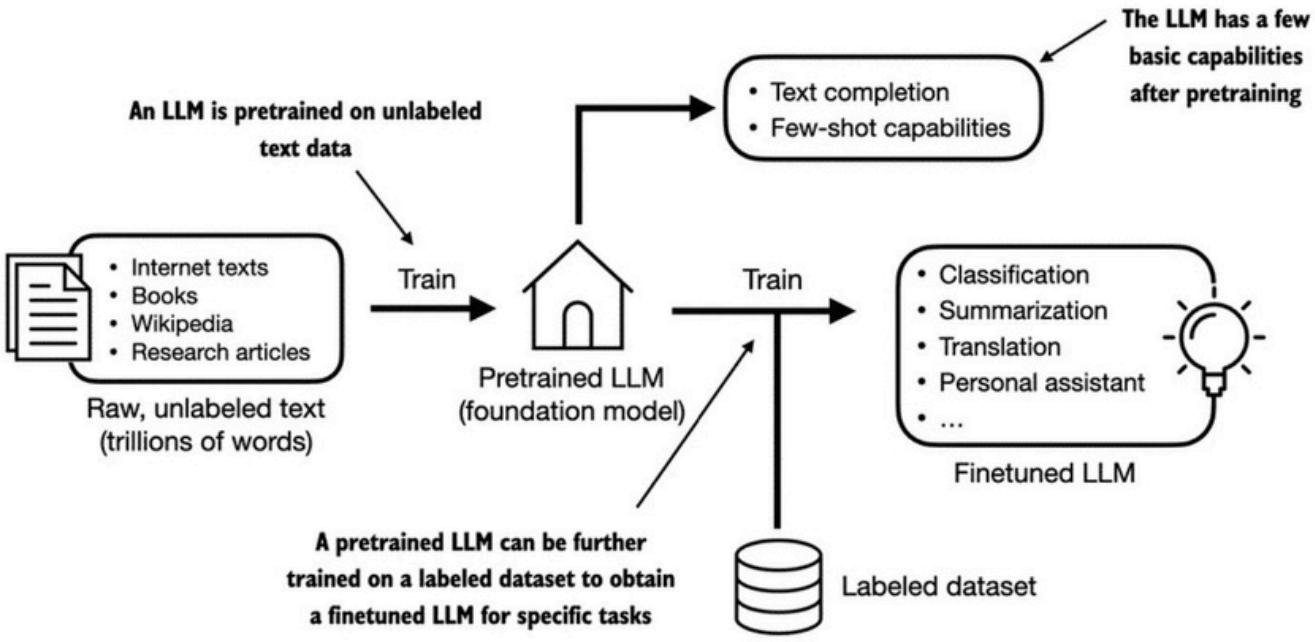
```
1.7 Building a large language model
1.8 Summary
```

AI including:

```
machine learning deep learning

rule-based systems genetic algorithms expert systems fuzzy logic symbolic
reasoning
```

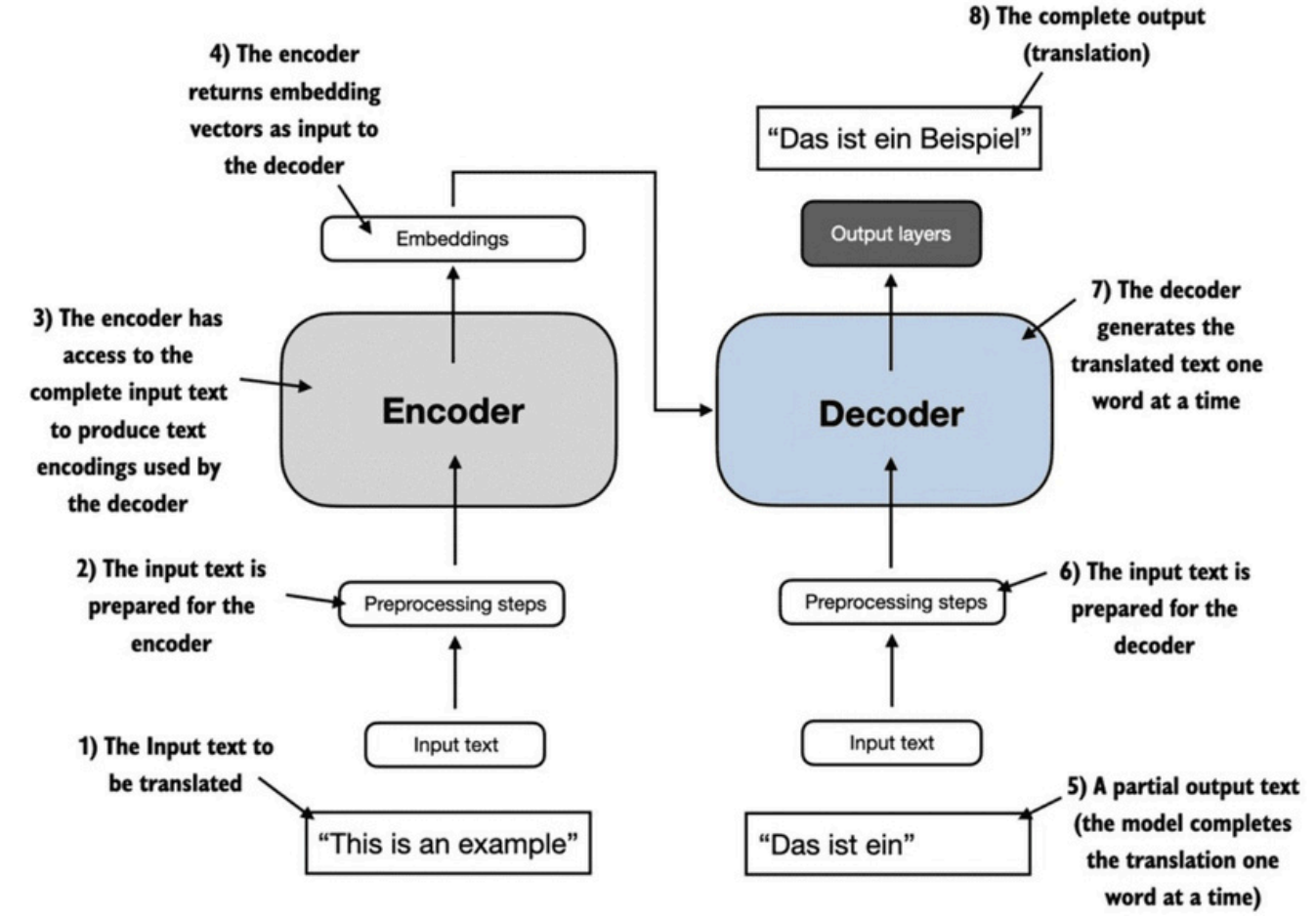
- Traditional machine learning: Human experts might manually extract features from email text such as the frequency of certain trigger words (“prize,” “win,” “free”), the number of exclamation marks, use of all uppercase words, or the presence of suspicious links.
- Deep learning: Does not require manual feature extraction. This means that human experts do not need to identify and select the most relevant features for a deep learning model.

> **Note:** Both traditional machine learning and deep learning require collecting data (e.g., which emails are spam, which are not). However, traditional machine learning also requires experts to manually extract features. Features for spam emails include: many words like "win", "free", suspicious links, usage of uppercase letters...

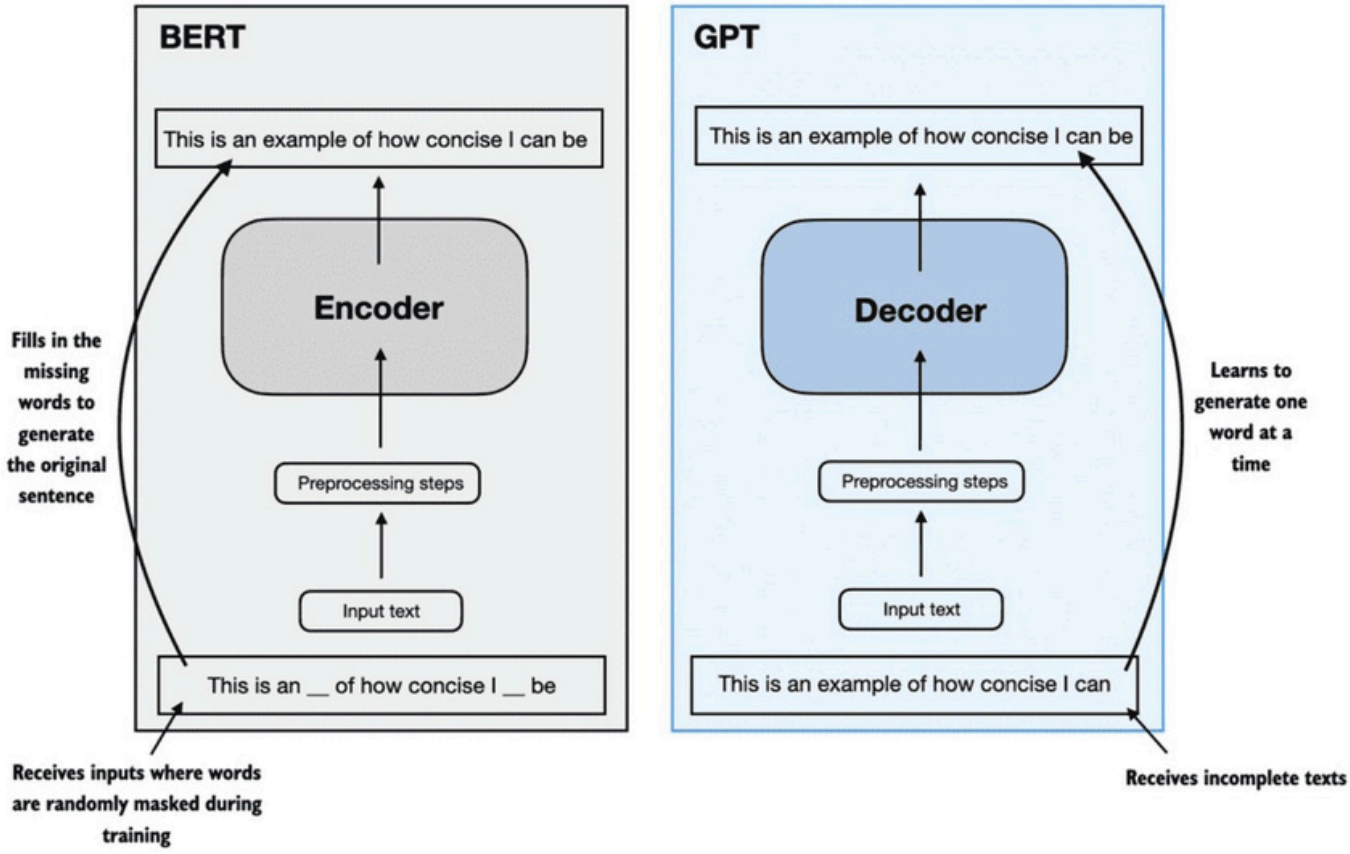


The two most popular categories of finetuning LLMs:

```
1. instruction-finetuning
2. finetuning for classification tasks
```

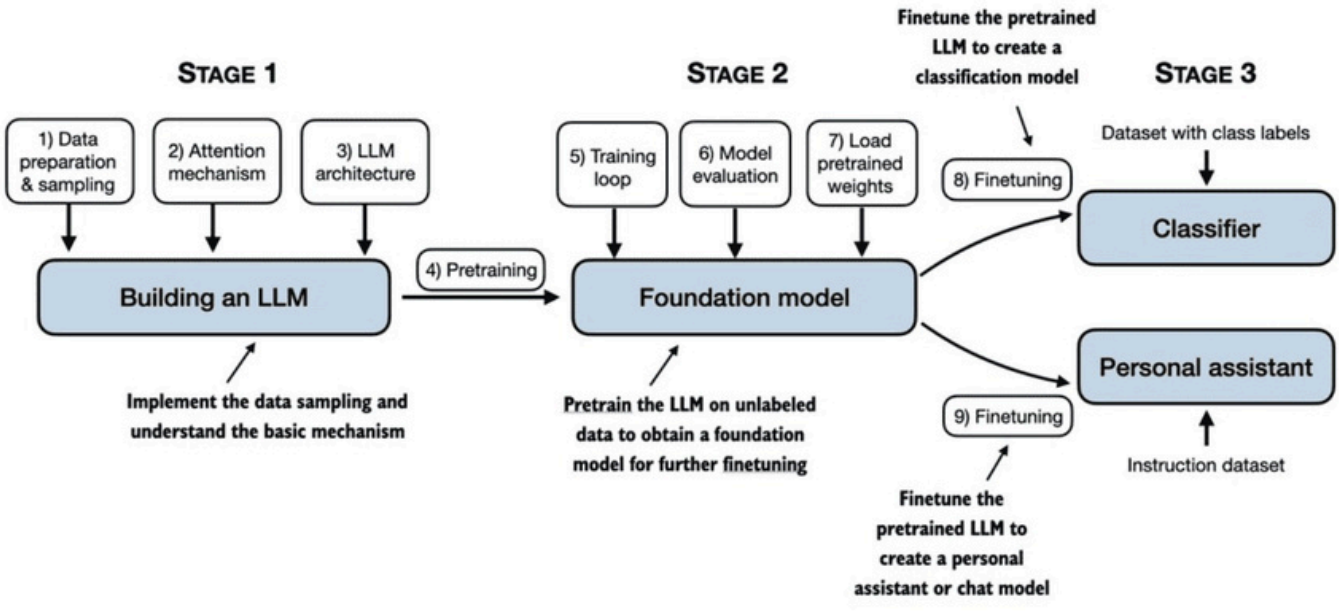


- A key component of transformers and LLMs is the self-attention mechanism, which allows the model to weigh the importance of different words or tokens in a sequence relative to each other.

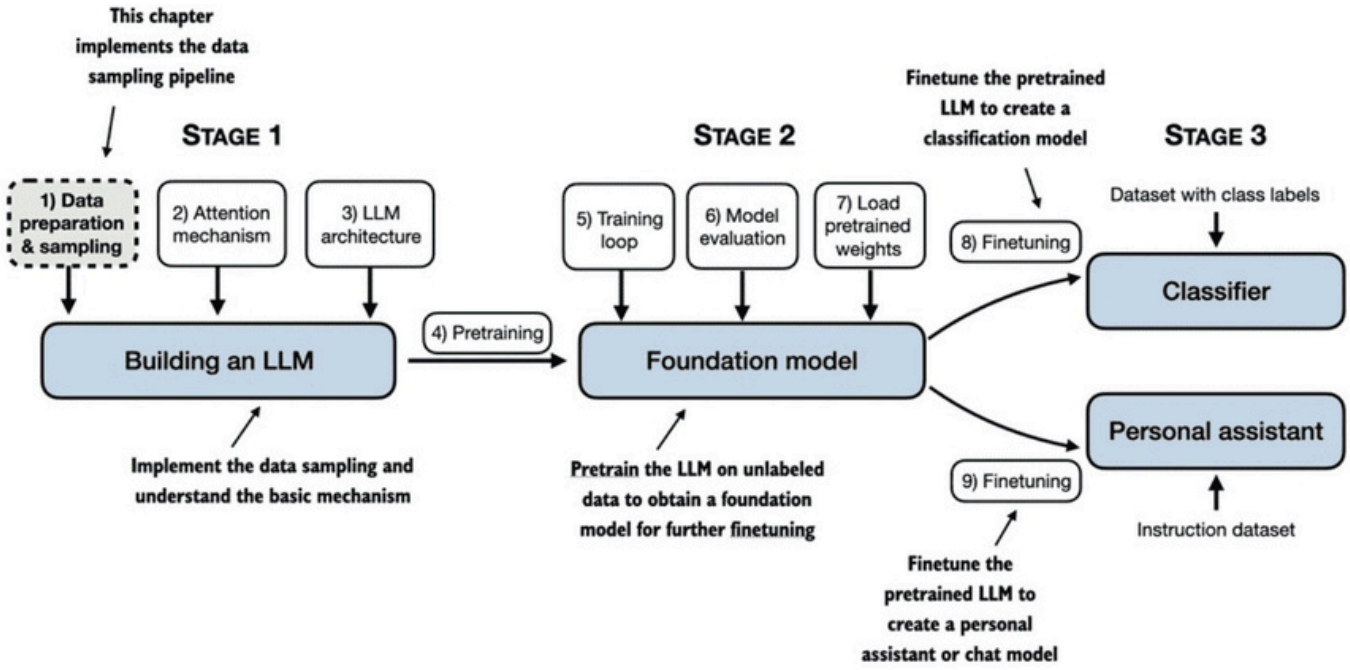


Dataset name	Dataset description	Number of tokens	Proportion in training data
CommonCrawl (filtered)	Web crawl data	410 billion	60%
WebText2	Web crawl data	19 billion	22%
Books1	Internet-based book corpus	12 billion	8%
Books2	Internet-based book corpus	55 billion	8%
Wikipedia	High-quality text	3 billion	3%

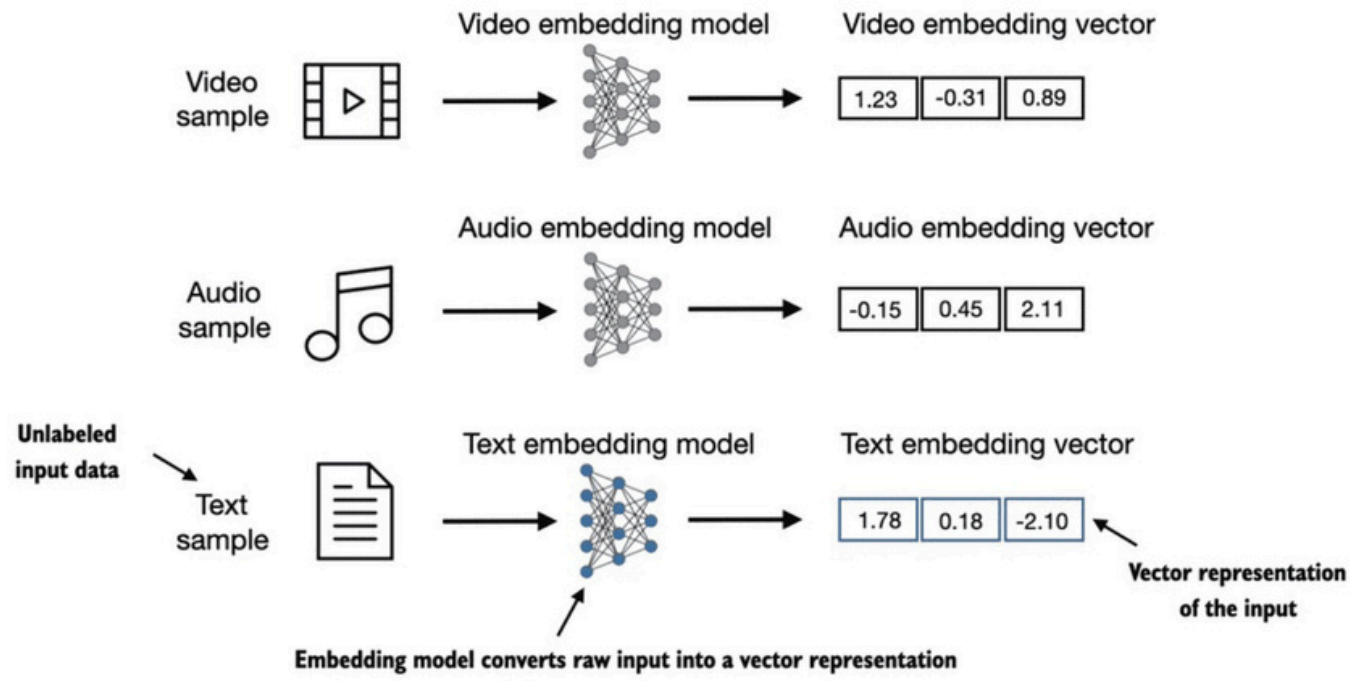
The ability to perform tasks that the model wasn't explicitly trained to perform is called an "emergent behavior" (涌现行为). "Emergent behavior" refers to abilities (like reasoning, arithmetic, etc.) that a model spontaneously exhibits as its size increases, which were not explicitly trained for. These abilities are not driven by individual parameters or specific tasks but are a spontaneous product of increased system scale and complexity.



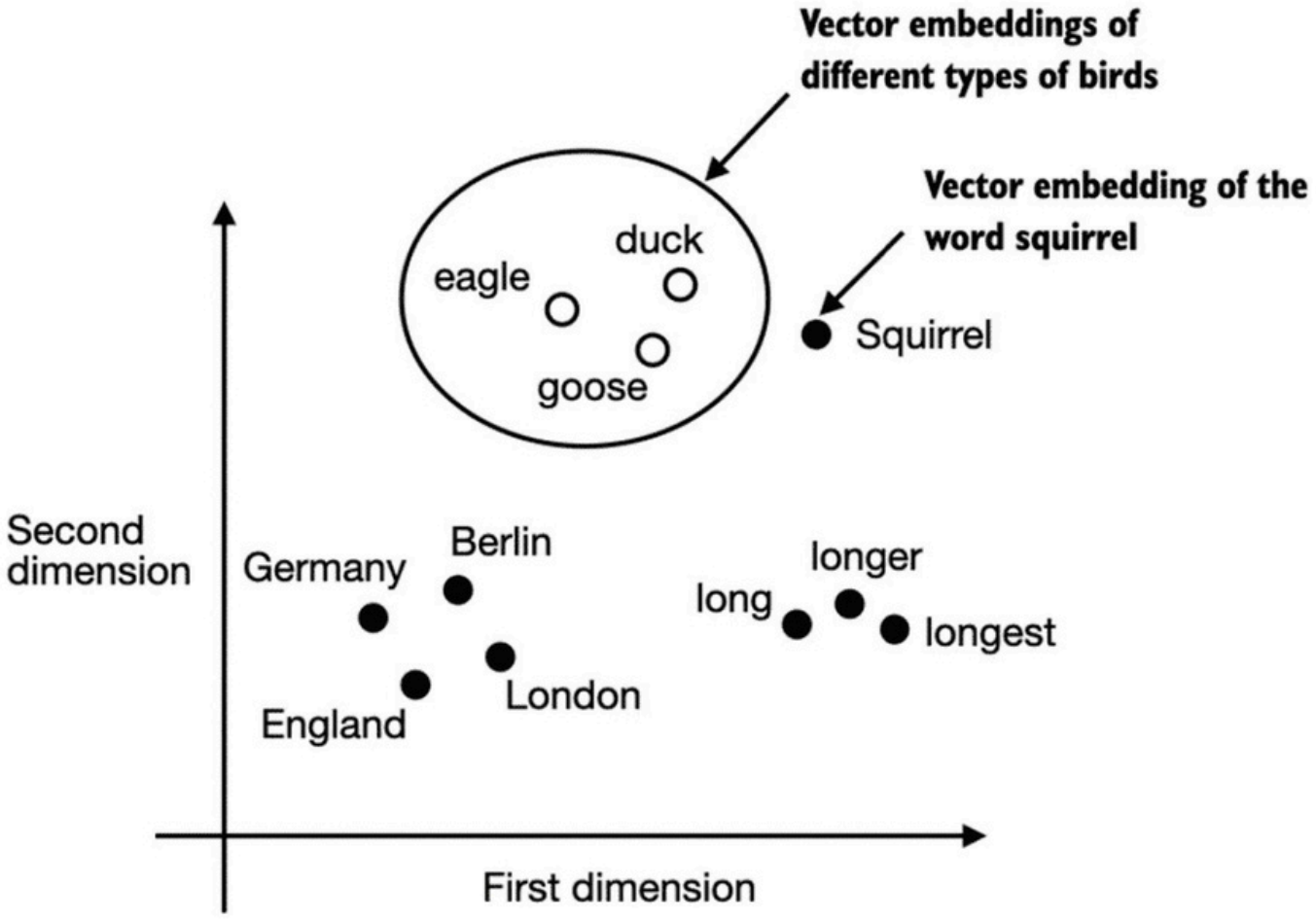
2. Working with Text Data



2.1 Understanding word embeddings

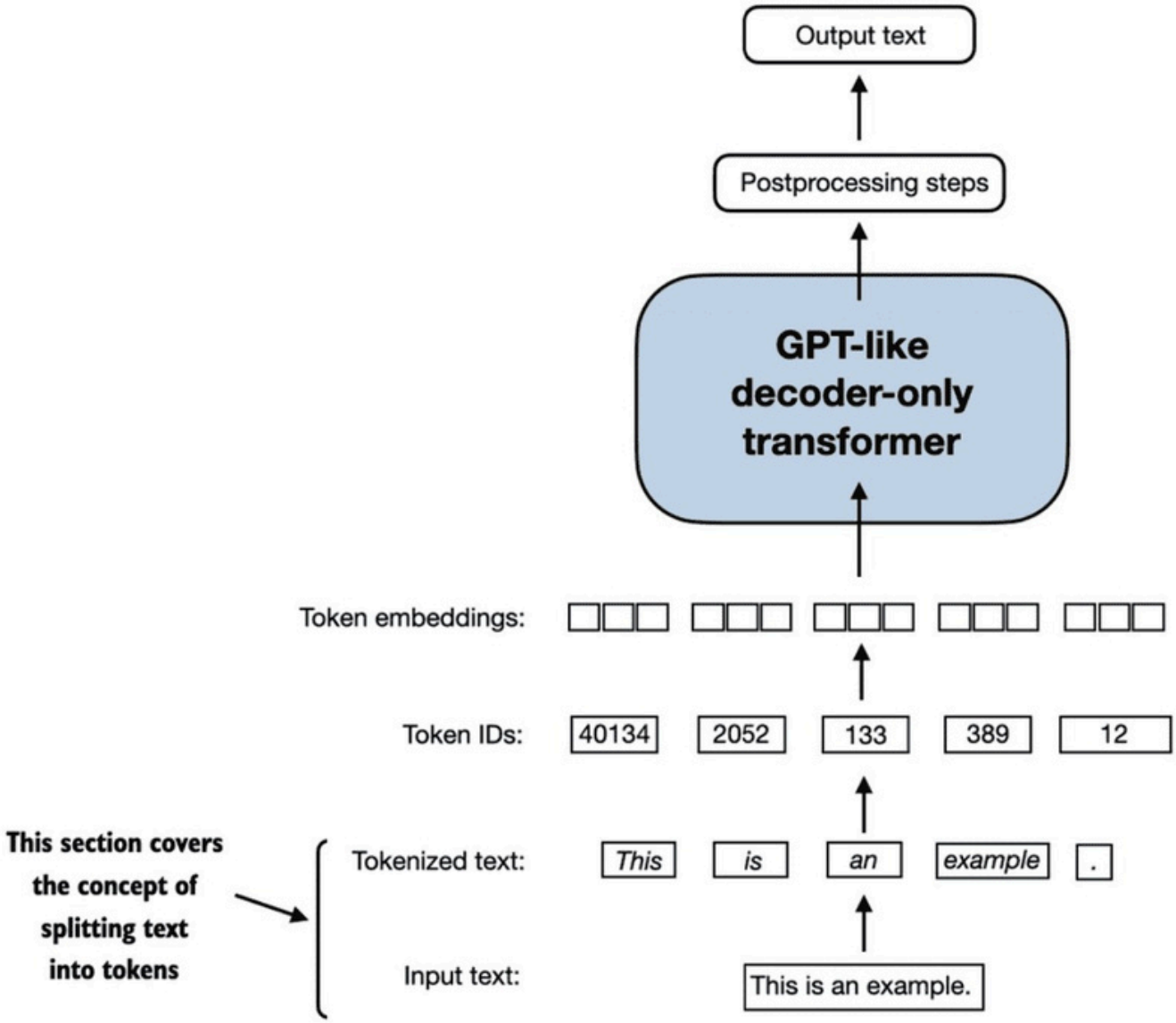


- While word embeddings are the most common form of text embedding, there are also embeddings for sentences, paragraphs, or whole documents. Sentence or paragraph embeddings are popular choices for retrieval- augmented generation.

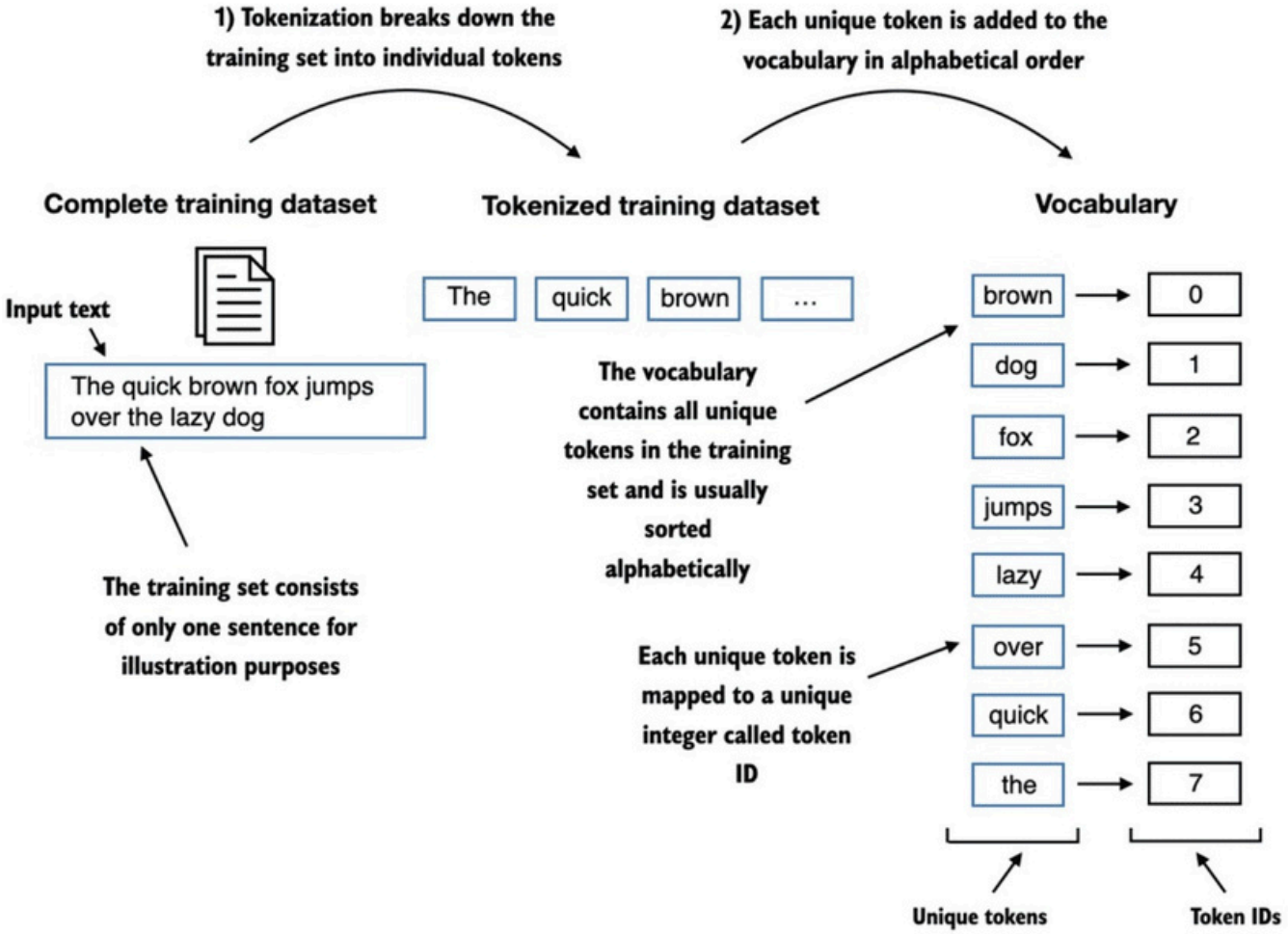


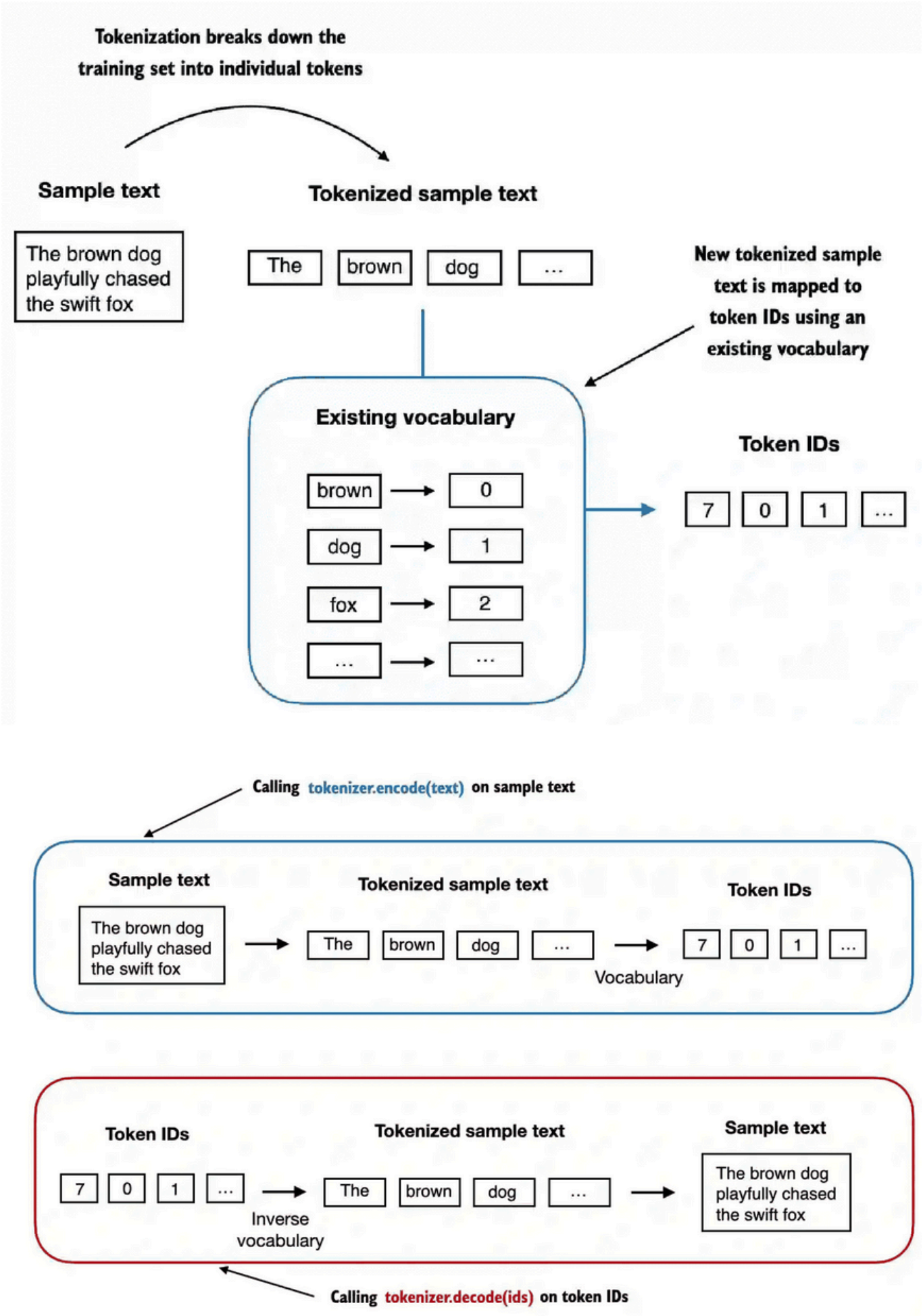
- Word embeddings can have varying dimensions, from one to thousands. As shown in Figure 2.3, we can choose two-dimensional word embeddings for visualization purposes.

2.2 Tokenizing text

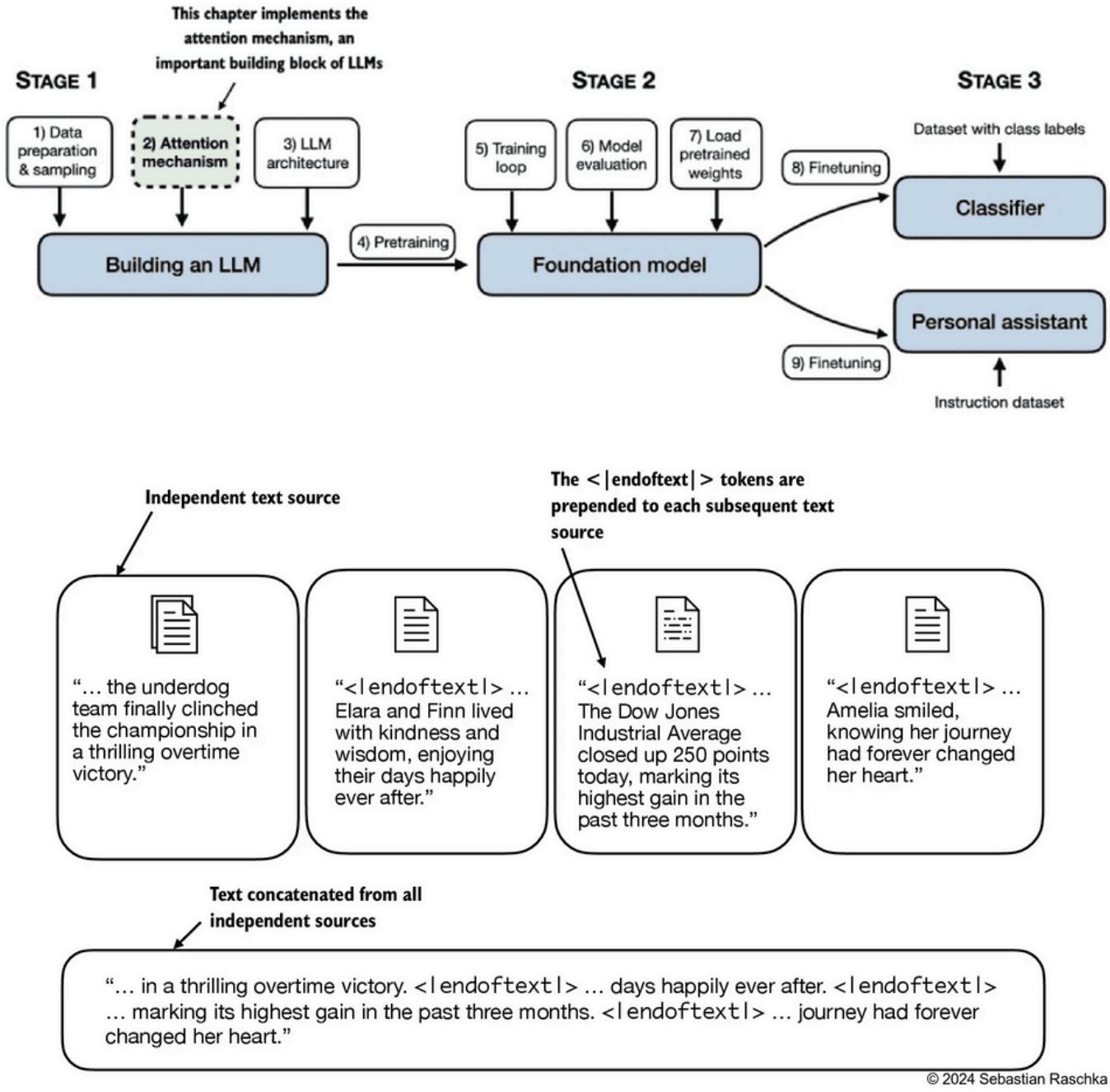


2.3 Converting tokens into token IDs


1) Tokenization breaks down the training set into individual tokens
2) Each unique token is added to the vocabulary in alphabetical order
Complete training dataset
Tokenized training dataset
Vocabulary
Input text
The quick brown fox jumps over the lazy dog
The training set consists of only one sentence for illustration purposes
The
quick
brown
...
The vocabulary contains all unique tokens in the training set and is usually sorted alphabetically
Each unique token is mapped to a unique integer called token ID
brown → 0
dog → 1
fox → 2
jumps → 3
lazy → 4
over → 5
quick → 6
the → 7
Unique tokens
Token IDs

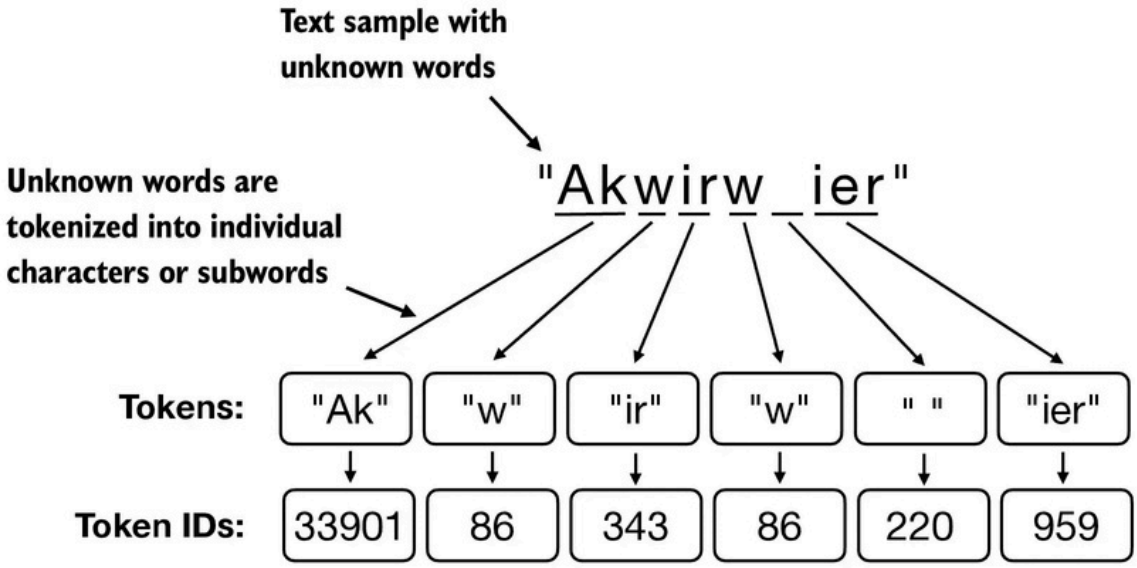


2.4 Adding special context tokens



Additional special tokens:

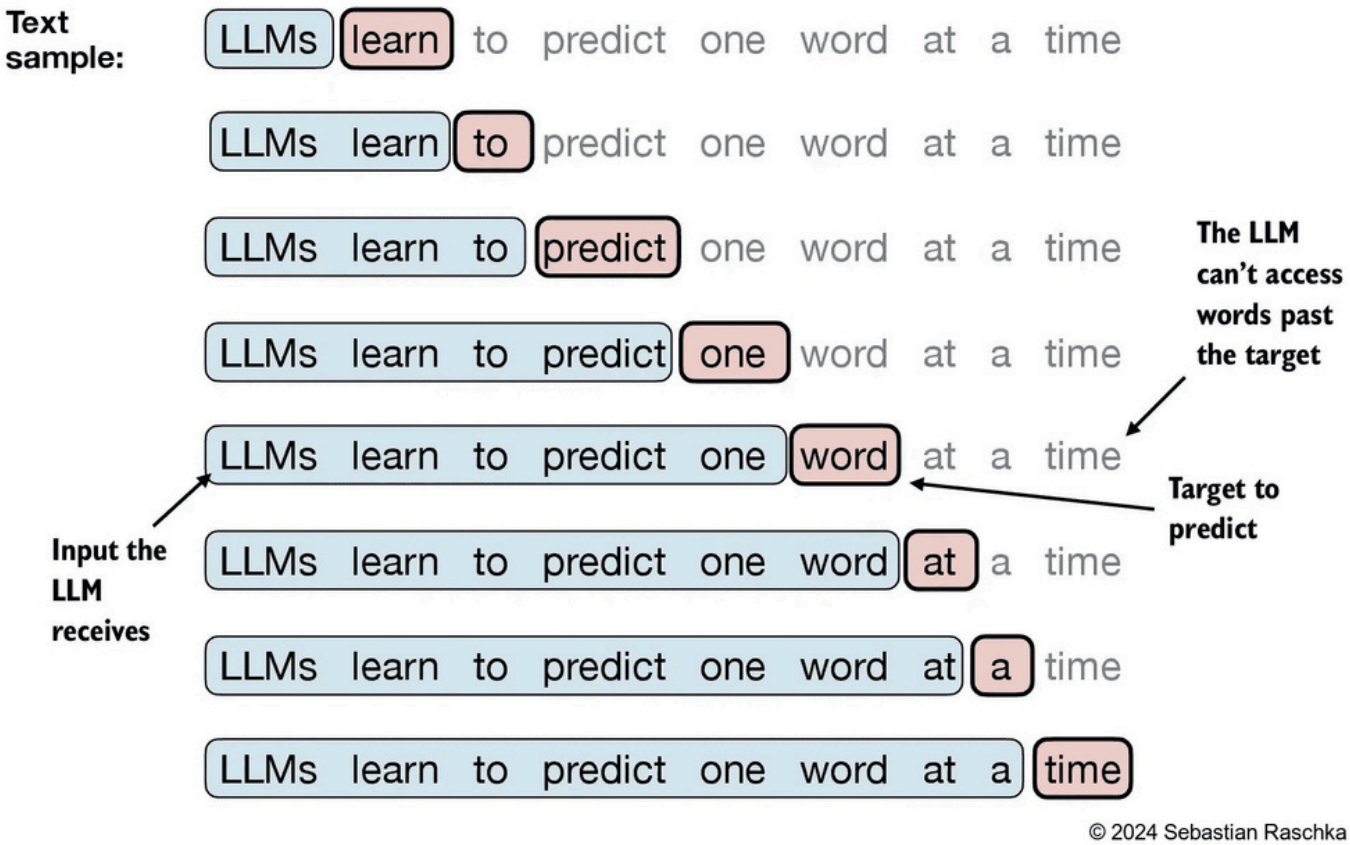
1. [BOS] (beginning of sequence)
2. [EOS] (end of sequence)
3. [PAD] (padding)
4. |endoftext|
5. |unk|



© 2024 Sebastian Raschka

- The original BPE tokenizer can be found here: https://github.com/openai/gpt-2/blob/master/src/encoder.py
- The BPE tokenizer from OpenAI's open-source tiktoken library, which implements its core algorithms in Rust to improve computational performance.

2.6 Data sampling with a sliding window

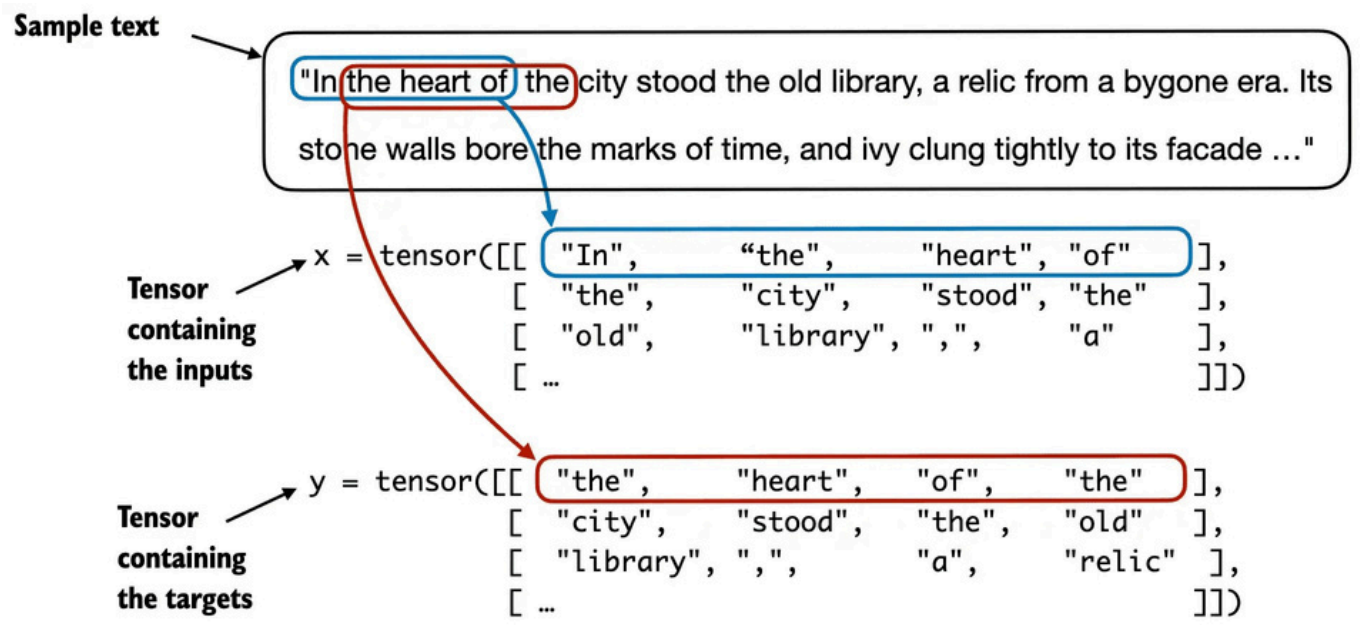


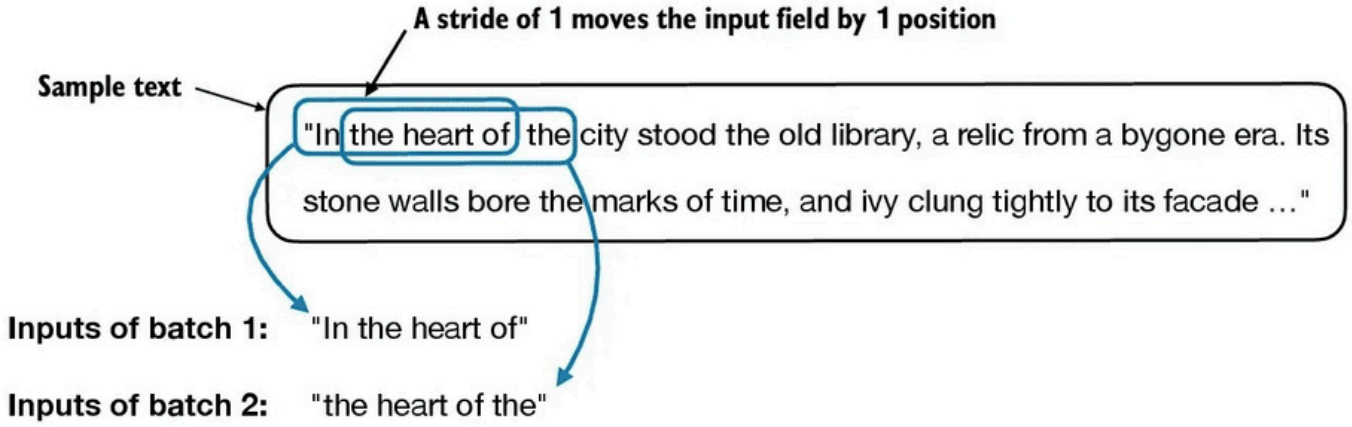
© 2024 Sebastian Raschka

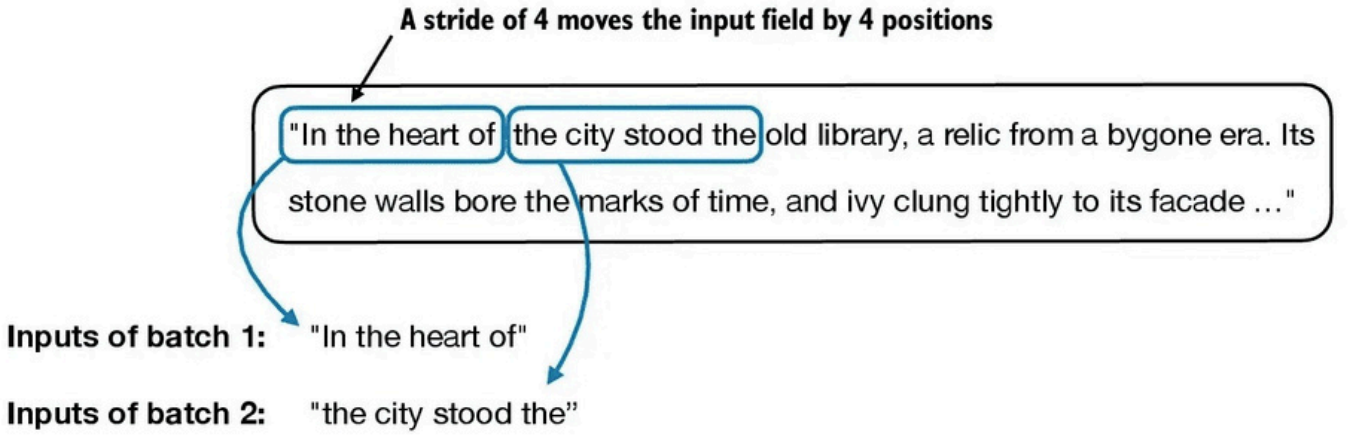
- For each text chunk, we want the inputs and targets
- Since we want the model to predict the next word, the targets are the inputs shifted by one position to the right

The prediction would look like as follows (input-target pairs):

```
and ----> established
and established ----> himself
and established himself ---->  in
and established himself in ---->  a
```







- Small batch sizes require less memory during training but lead to more noisy model updates. The batch size is a trade-off and hyperparameter to experiment with when training LLMs.
- Example (batch_size=1, max_length=4, stride=1):

```
[tensor([[  40,  367, 2885, 1464]]), tensor([[ 367, 2885, 1464, 1807]])]
```

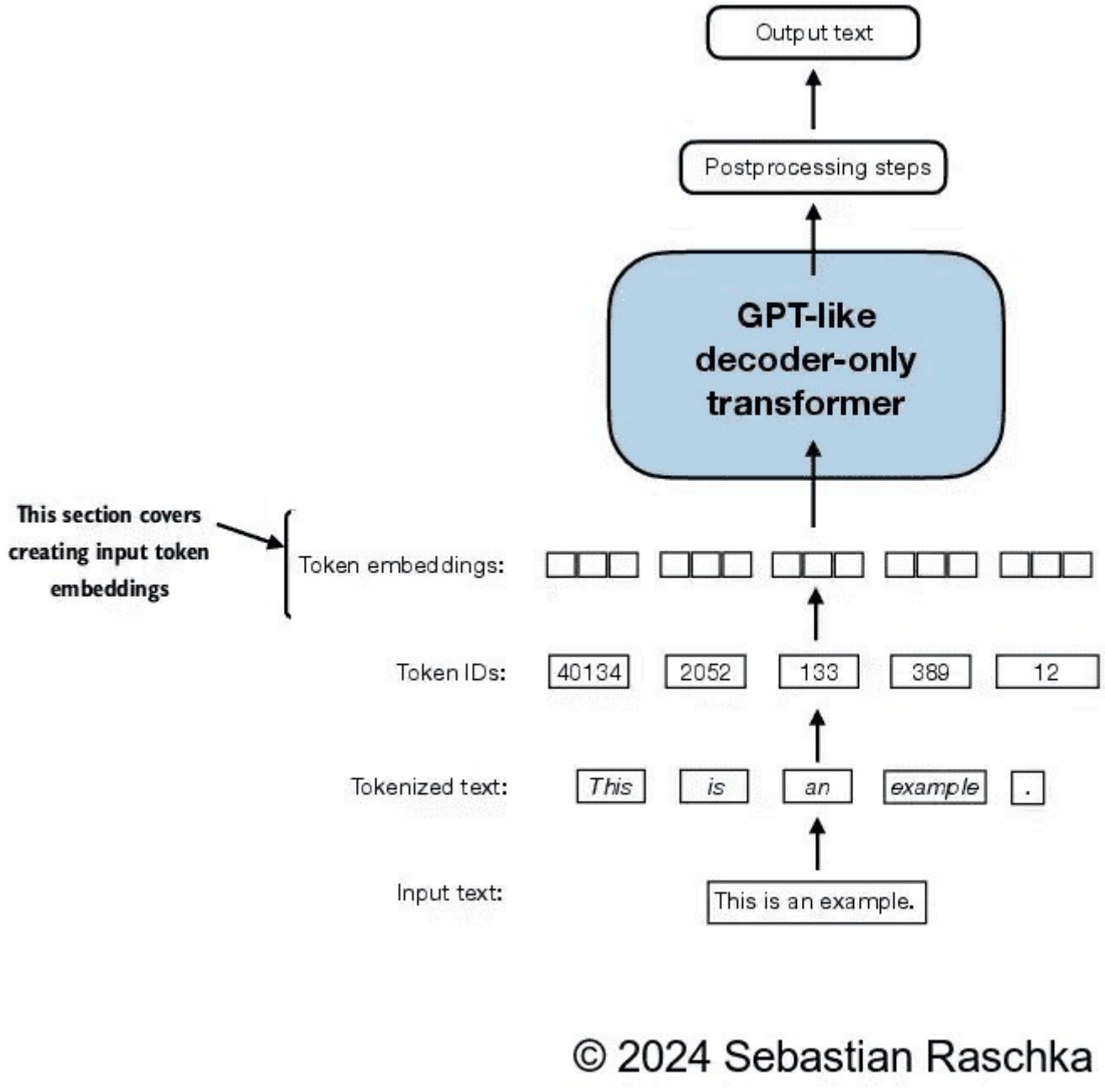
- Example (batch_size=8, max_length=4, stride=4):

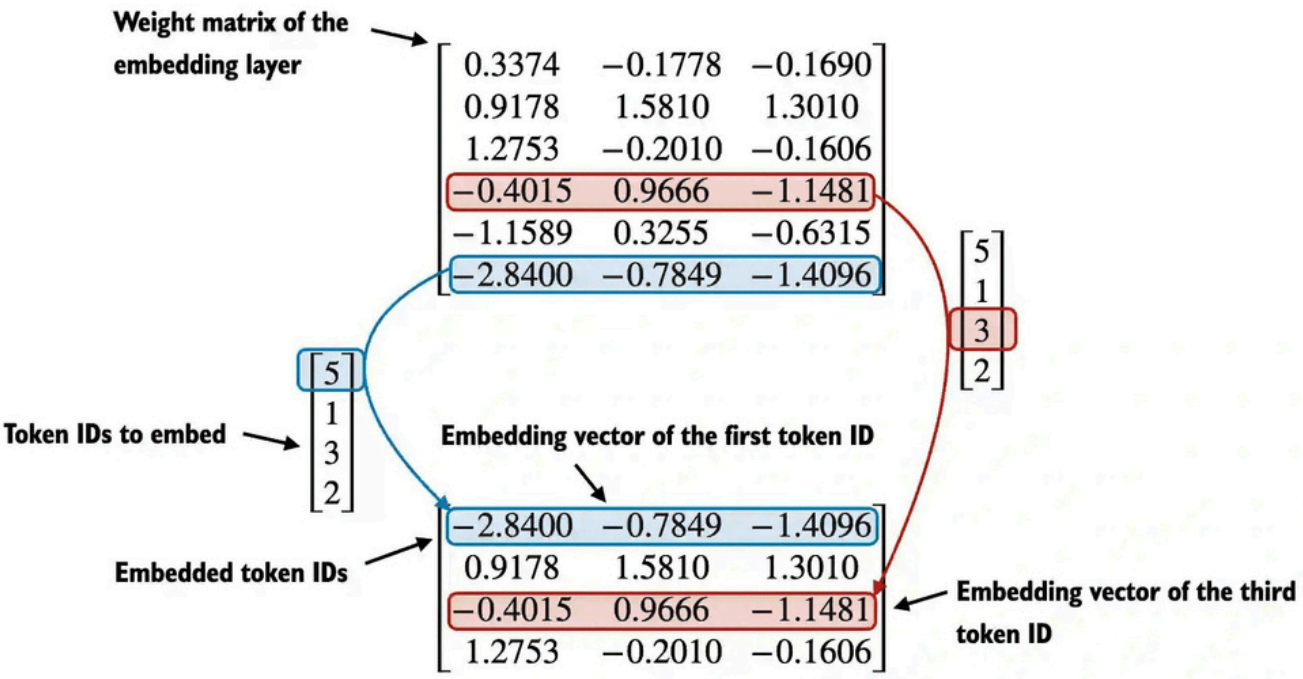
```
Inputs:
 tensor([[   40,   367, 2885, 1464],
        [  1807,  3619,  402,   271],
        [10899, 2138,     257, 7026],
        [15632, [ 438,  2016,  257],
        1576, 373, 284, 326,    438],
        [ 568, [ 340,           645],
        5975,   [   284,        502],
        3285,                    11]])

Targets:

 tensor([[ 367, 2885, 1464, 1807],
```

```
        [ 3619,   402,    271, 10899],
        [ 2138,   257,   7026, 15632],
        [  438,  2016,   257,    922],
        [ 5891,  1576,   438,    568],
        [  340,   373,     645, 1049],
        [ 5975,   284,    502,   284],
        [ 3285,   326,     11,   287]])
```

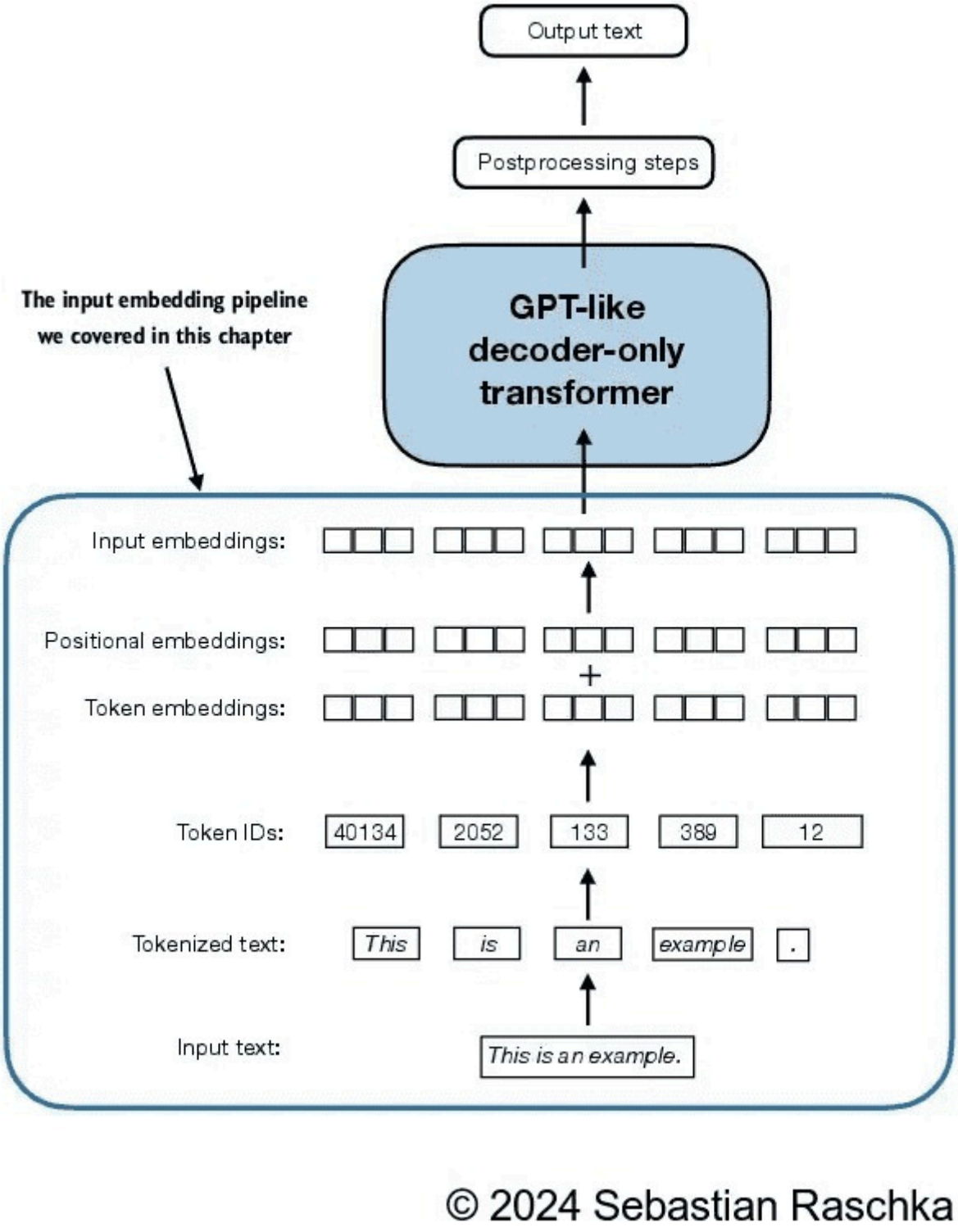
2.7 Creating token embeddings

- We converted the token IDs into a continuous vector representation, the so-called token embeddings.





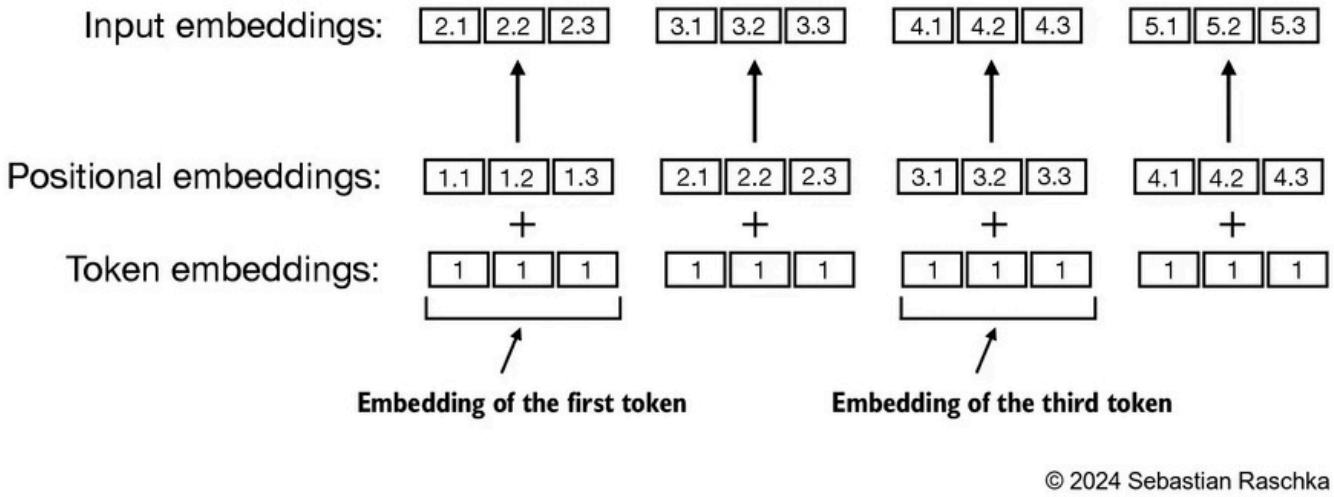
2.8 Encoding word positions



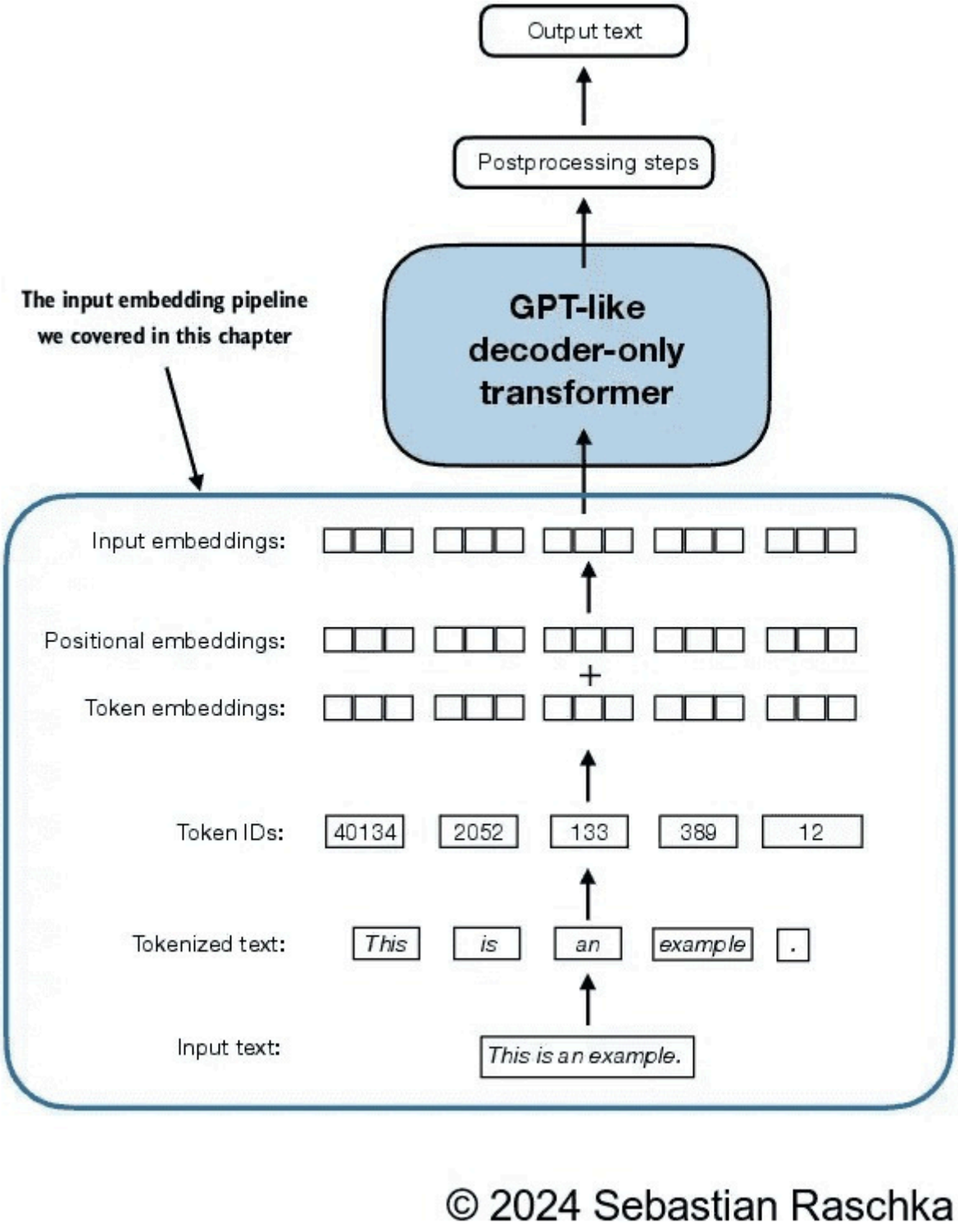
- In principle, the deterministic, position-independent embedding of the token ID is good for reproducibility purposes.
- However, since the self-attention mechanism of LLMs itself is also position-agnostic, it is helpful to inject additional position information into the LLM.

Two broad categories of position-aware embeddings:

```
1. relative positional embeddings
2. absolute positional embeddings
```



- Instead of focusing on the absolute position of a token, the emphasis of relative positional embeddings is on the relative position or distance between tokens. This means the model learns the relationships in terms of "how far apart" rather than "at which exact position." The advantage here is that the model can generalize better to sequences of varying lengths, even if it hasn't seen such lengths during training.
- OpenAI's GPT models use absolute positional embeddings that are optimized during the training process rather than being fixed or predefined like the positional encodings in the original Transformer model. This optimization process is part of the model training itself, which we will implement later in this book. For now, let's create the initial positional embeddings to create the LLM inputs for the upcoming chapters.



- `context_length` is a variable that represents the supported input size of the LLM.

2.9 Summary

- LLMs require textual data to be converted into numerical vectors, known as embeddings since they can't process raw text. Embeddings transform discrete data (like words or images) into continuous vector spaces, making them compatible with neural network operations.
- As the first step, raw text is broken into tokens, which can be words or characters. Then, the tokens are converted into integer representations, termed token IDs.
- Special tokens, such as <|unk|> and <|endoftext|>, can be added to enhance the model's understanding and handle various contexts, such as unknown words or marking the boundary between