# Transformer & GPT

# For NLP Problems

- **Transformer**： initially targeted at natural language processing (NLP)
  - The network input : a series of high-dimensional embeddings representing words or word fragments.
  - Language datasets share some of the characteristics of image data.
  - The number of input variables can be very large, and the statistics are similar at every position;
  - it’s not sensible to re-learn the meaning of the word dog at every possible position in a body of text.
  - However, language datasets have the complication that **text sequences vary in length**, and unlike images, there is no easy way to resize them.

# Processing text data

Term: **transformers**, the former word should pay attention to the latter.

- **connections between the words** and strength of these connections will depend on the words themselves.
- These connections need to extend across large text spans.

  - The restaurant refused to serve me a ham sandwich because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambiance was just as good as the food and service.

# Dot-product self-attention

- Models for processing text should satisfy
  - to cope with long input passages of **differing lengths**
  - to contain **connections** between word representations
  - The transformer acquires both properties by using ***dot-product self-attention*.**

$$f[x] = \mathrm{ReLU}[Wx + b],$$

  - where $b$ is the bias, $W$ is the connection weights

# Dot-product self-attention

**Self-Attention Block** $\mathbf{sa}[\bullet]$

- N inputs $x_1, \cdots, x_N$ embedded in a vector in $R^D$

$$v_m = Wx_m + b$$

- The *n*-th output of $\mathbf{sa}[x_1, \cdots, x_N]$

$$\mathbf{sa}_n[x_1, \cdots, x_N] = \sum_{m=1}^{N} a[x_m, x_n]\, v_m$$

- $a[x_m, x_n]$ is the *attention* that the *n-th* output pays to input $\mathbf{x}_m$ .
- The *N* weights $a[x_m, x_n]$ are non-negative and sum to one
- Self-attention can be thought of as *routing* the values in different proportions to create each output

# Dot-product self attention

To compute the attention, two more linear transformations to the inputs:

$$\mathbf{q}_n = \beta_q + \Omega_q X_n$$

$$\mathbf{k}_n = \beta_k + \Omega_k X_n$$

where $\mathbf{q}_n$ and $\mathbf{k}_n$ are termed ***queries*** and ***keys***, respectively. Then we compute dot products between the queries and keys

$$a[\mathbf{x}_m, \mathbf{x}_n] = \mathrm{softmax}_m[\mathbf{k}_\bullet^T \mathbf{q}_n]$$

$$= \frac{\exp[\mathbf{k}_m^T \mathbf{q}_n]}{\sum_{j=1}^{N} \exp[\mathbf{k}_j^T \mathbf{q}_n]}$$

so for each **x***n*, they are positive and sum to one

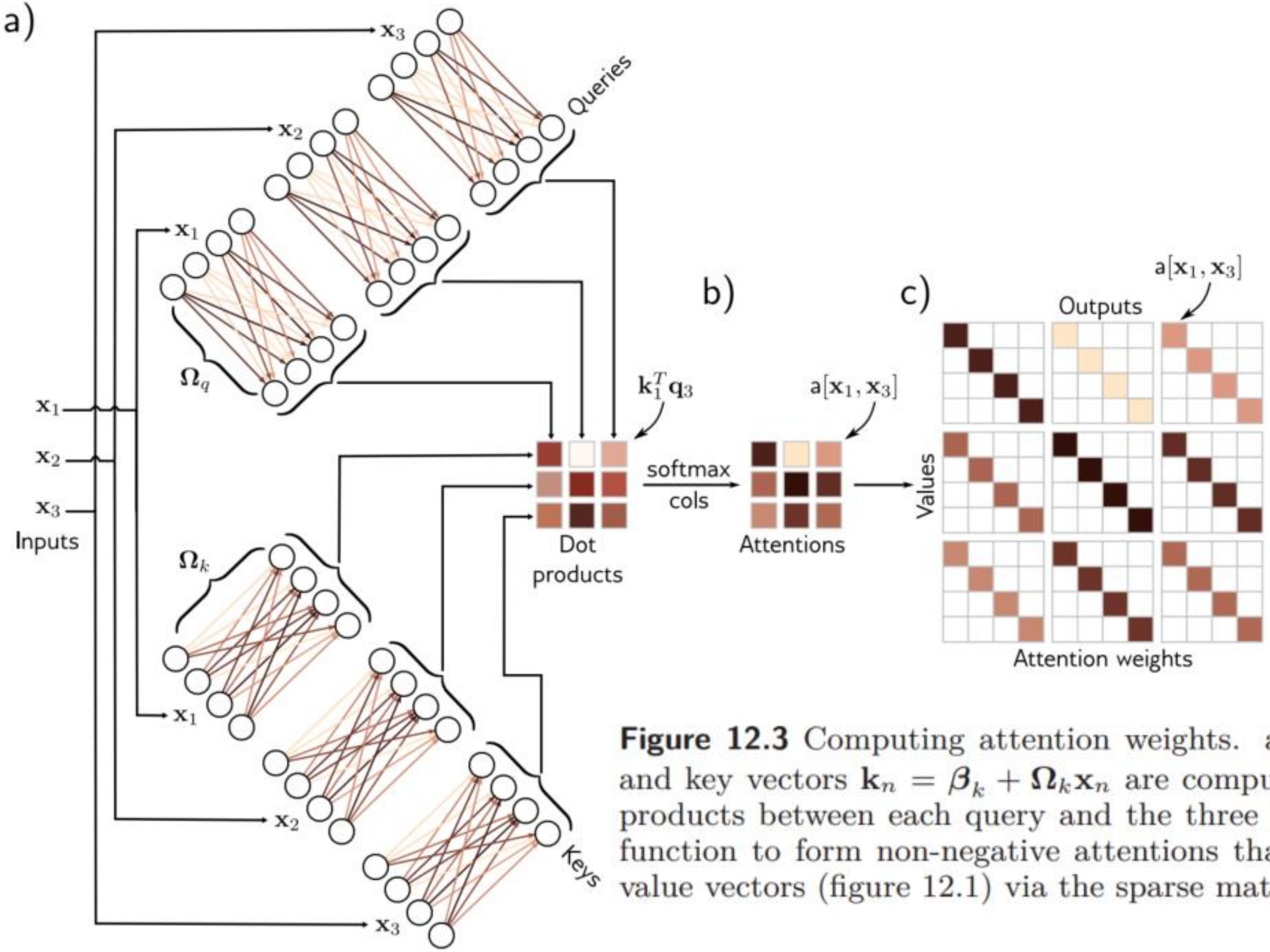
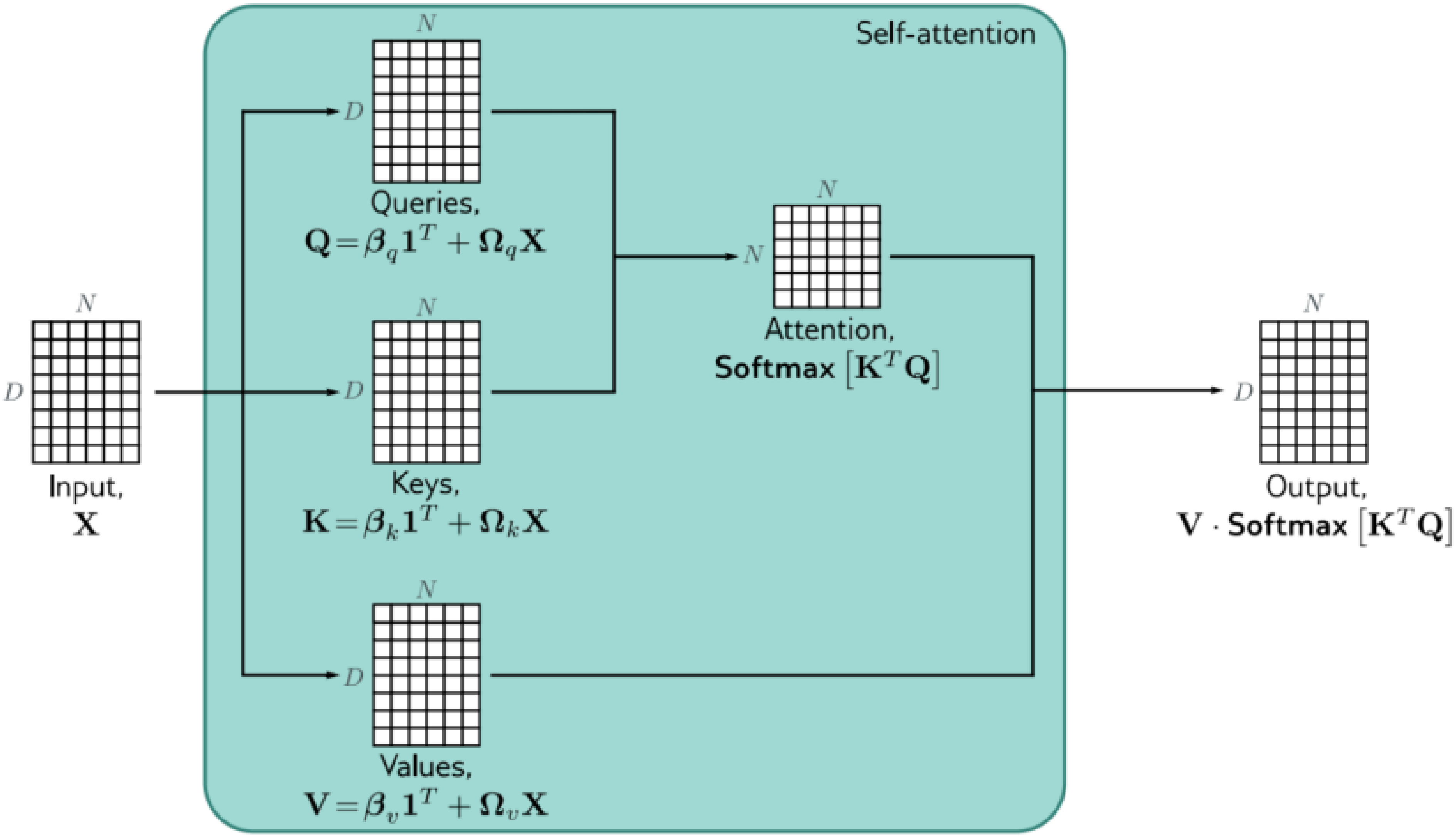



**Figure 12.3** Computing attention weights. a) Query vectors $\mathbf{q}_n = \boldsymbol{\beta}_q + \mathbf{\Omega}_q\mathbf{x}_n$ and key vectors $\mathbf{k}_n = \boldsymbol{\beta}_k + \mathbf{\Omega}_k\mathbf{x}_n$ are computed for each input $\mathbf{x}_n$. b) The dot products between each query and the three keys are passed through a softmax function to form non-negative attentions that sum to one. c) These route the value vectors (figure 12.1) via the sparse matrix from figure 12.2c.

# Advantages: Attention is great

- **Attention significantly improves performance**
  - It's very useful to allow decoder to focus on certain parts of the source
- **Attention solves the bottleneck problem**
  - Attention allows decoder to look directly at source; bypass bottleneck
- **Attention helps with vanishing gradient problem**
  - Provides shortcut to faraway states
- **Attention provides some interpretability**
  - By inspecting attention distribution, we can see what the decoder was focusing on

# Self-attention summary



# Matrix Form

### Compact form of Self-attention

- if the N inputs xn form the columns of the D × N matrix X. The values, queries, and keys can be computed as:

$$\mathbf{V}[\mathbf{X}] = \beta_v \mathbf{1}^T + \Omega_v \mathbf{X},$$
$$\mathbf{Q}[\mathbf{X}] = \beta_q \mathbf{1}^T + \Omega_q \mathbf{X},$$
$$\mathbf{K}[\mathbf{X}] = \beta_k \mathbf{1}^T + \Omega_k \mathbf{X}$$

where **1** is an $N \times 1$ vector containing ones. The self-attention computation is rewritten as the following compact form

$$\mathrm{Sa}[\mathbf{X}] = \mathbf{V}[\mathbf{X}] \cdot \mathrm{Softmax}\left[\left[\mathbf{K}[\mathbf{X}]^T \mathbf{Q}[\mathbf{X}]\right]\right],$$

where the function **Softmax**[•] performs the softmax operation independently on each of its columns

# Positional encoding

- **The self-attention mechanism discards order information:**
  - Ignore the order of the inputs $\mathbf{x}_n$. More precisely, it is equivariant with respect to input permutations.
- **Absolute positional encodings & Relative positional encodings**
  - Each column of $\mathbf{\Pi}$ is unique and hence contains information about the absolute position in the input sequence.
  - Each element of the attention matrix corresponds to a particular offset $\pi_{a,b}$ between **query position** $\boldsymbol{a}$ and **key position** $\boldsymbol{b}$.

# Scaled dot product self-attention

- Problem: Large components dominate;
  - Small changes to the inputs to the softmax function now have little effect on the output (i.e., the gradients are very small), making the model difficult to train.
- Solution: dot products are scaled by the square root of the dimension $D_q$ of the queries and keys :

$$\mathrm{Sa}[\mathbf{X}] = \mathbf{V}[\mathbf{X}] \cdot \mathrm{Softmax}\left[\frac{\mathbf{K}[\mathbf{X}]^T \mathbf{Q}[\mathbf{X}]}{\sqrt{D_q}}\right],$$

- This is known as scaled dot product self-attention.

# Multiple heads

Multiple self-attention mechanisms are usually applied in parallel, and this is known as *multi-head self-attention*.

➢ *H* different sets of values, keys, and queries are computed:

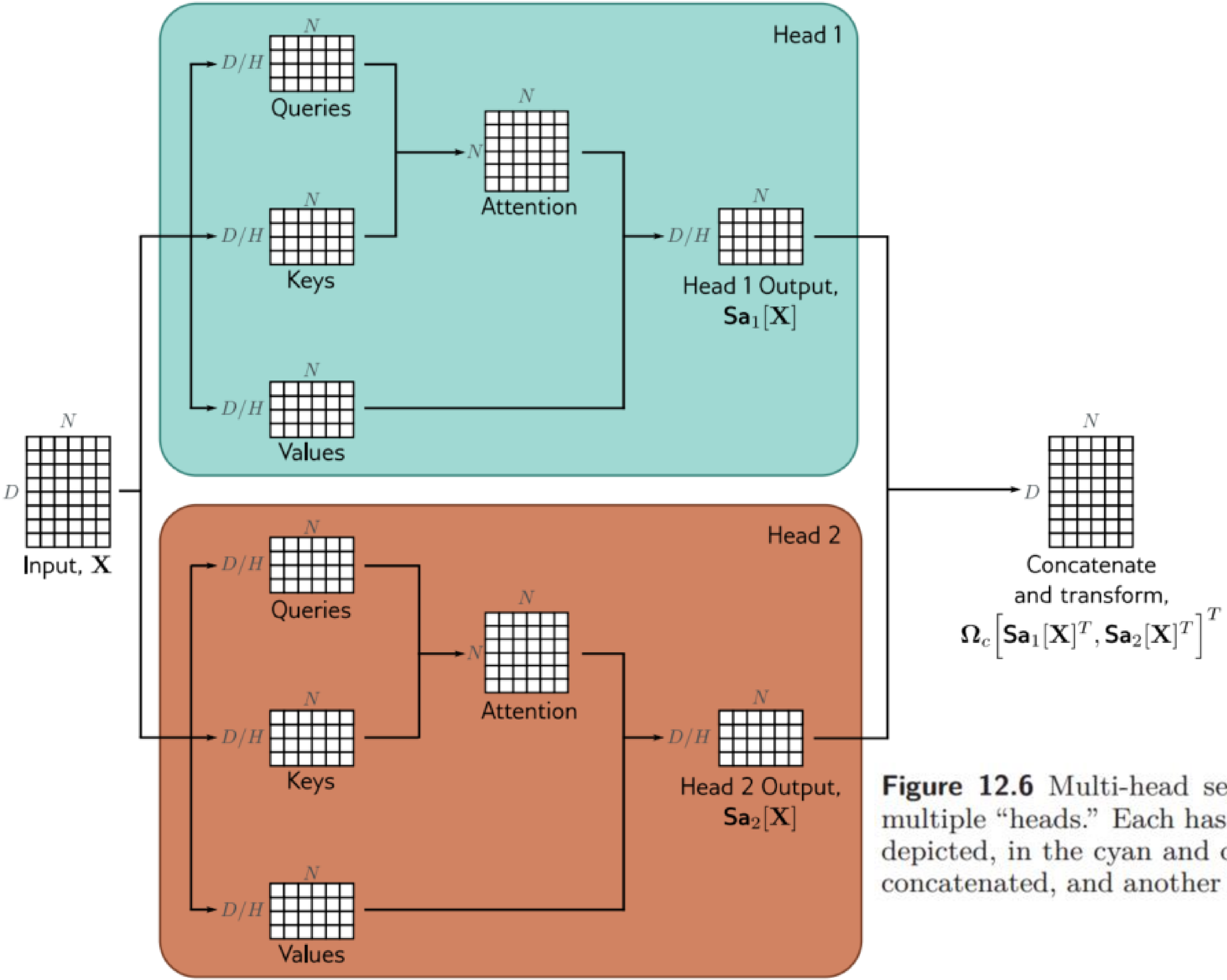
$$\begin{aligned}\mathbf{V}_h &= \beta_{vh}\mathbf{1}^T + \Omega_{vh}\mathbf{X},\\ \mathbf{Q}_h &= \beta_{qh}\mathbf{1}^T + \Omega_{qh}\mathbf{X},\\ \mathbf{K}_h &= \beta_{kh}\mathbf{1}^T + \Omega_{kh}\mathbf{X}\end{aligned}$$

The $\mathrm{h}^{\mathrm{th}}$ self-attention mechanism or **head** can be written as:

$$\mathrm{Sa}_h[\mathbf{X}] = \mathbf{V}_h \cdot \mathrm{Softmax}\left[\frac{\mathbf{K}_h^T\mathbf{Q}_h}{\sqrt{D_q}}\right],$$

where parameter set $\{\beta_{vh}, \Omega_{vh}\}, \{\beta_{qh}, \Omega_{qh},\}$ and $\{\beta_{kh}, \Omega_{kh}\}$ represents a **head**.

$$\mathrm{MhSa}[\mathbf{X}] = \Omega_c\left[\mathrm{Sa}_1[\mathbf{X}]^T, \mathrm{Sa}_2[\mathbf{X}]^T, \cdots, \mathrm{Sa}_H[\mathbf{X}]^T\right]^T$$




$$\mathbf{X} \in R^{D\times N},$$

$$\mathbf{V}_h, \mathbf{Q}_h, \mathbf{K}_h \in R^{(D/H)\times N}$$

**Figure 12.6** Multi-head self-attention. Self-attention occurs in parallel across multiple "heads." Each has its own queries, keys, and values. Here two heads are depicted, in the cyan and orange boxes, respectively. The outputs are vertically concatenated, and another linear transformation $\mathbf{\Omega}_c$ is used to recombine them.

# Transformers

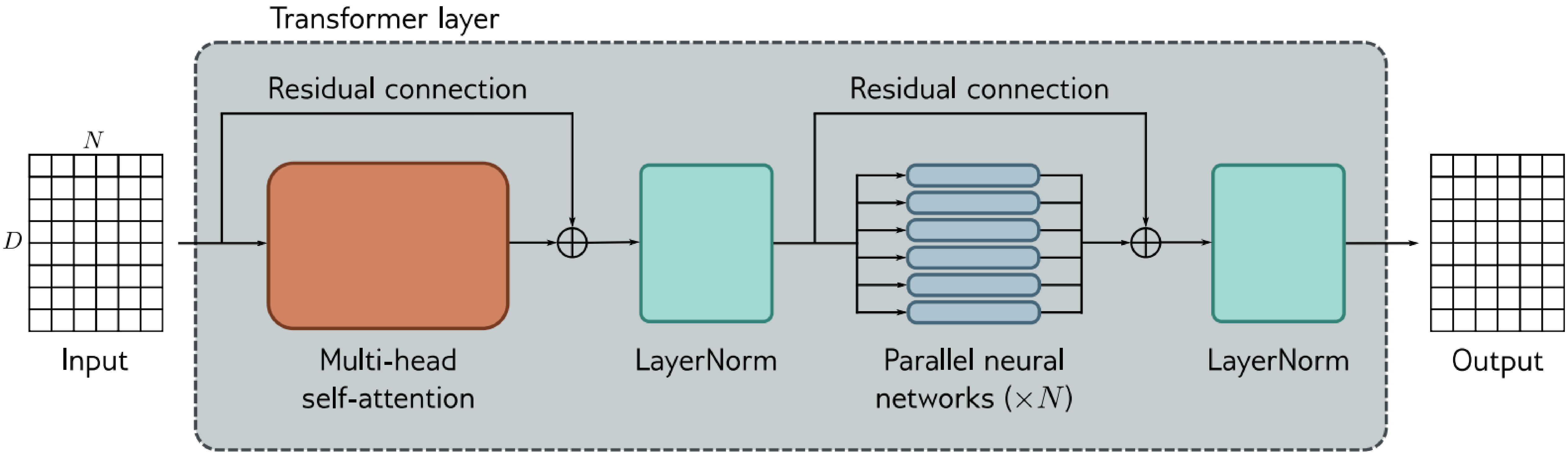
A transformer consists of a multi-head self-attention unit + a fully connected network $\mathbf{mlp}[\mathbf{x}\bullet]$. Both units are residual networks. In addition, it is typical to add a LayerNorm operation after both the self-attention and fully connected networks.

$$
\begin{aligned}
\mathbf{X} &\leftarrow \mathbf{X} + \mathrm{MhSa}[\mathbf{X}], \\
\mathbf{X} &\leftarrow \mathrm{LayerNorm}[\mathbf{X}] \\
\mathbf{x}_n &\leftarrow \mathbf{x}_n + \mathrm{mlp}[\mathbf{x}_n], \qquad \forall n \in \{1, 2, \cdots, N\} \\
\mathbf{X} &\leftarrow \mathrm{LayerNorm}[\mathbf{X}]
\end{aligned}
$$

where the column vectors $\mathbf{x}_n$ is the n-th column of $\mathbf{X}$

In a real network, the data passes through a series of these transformers.

# The structure of Transformer




- a multi-head attention block with a residual block, + LayerNorm operation
- A fully connected neural network with second residual block + LayerNorm operation

# Tokenization & Embedding

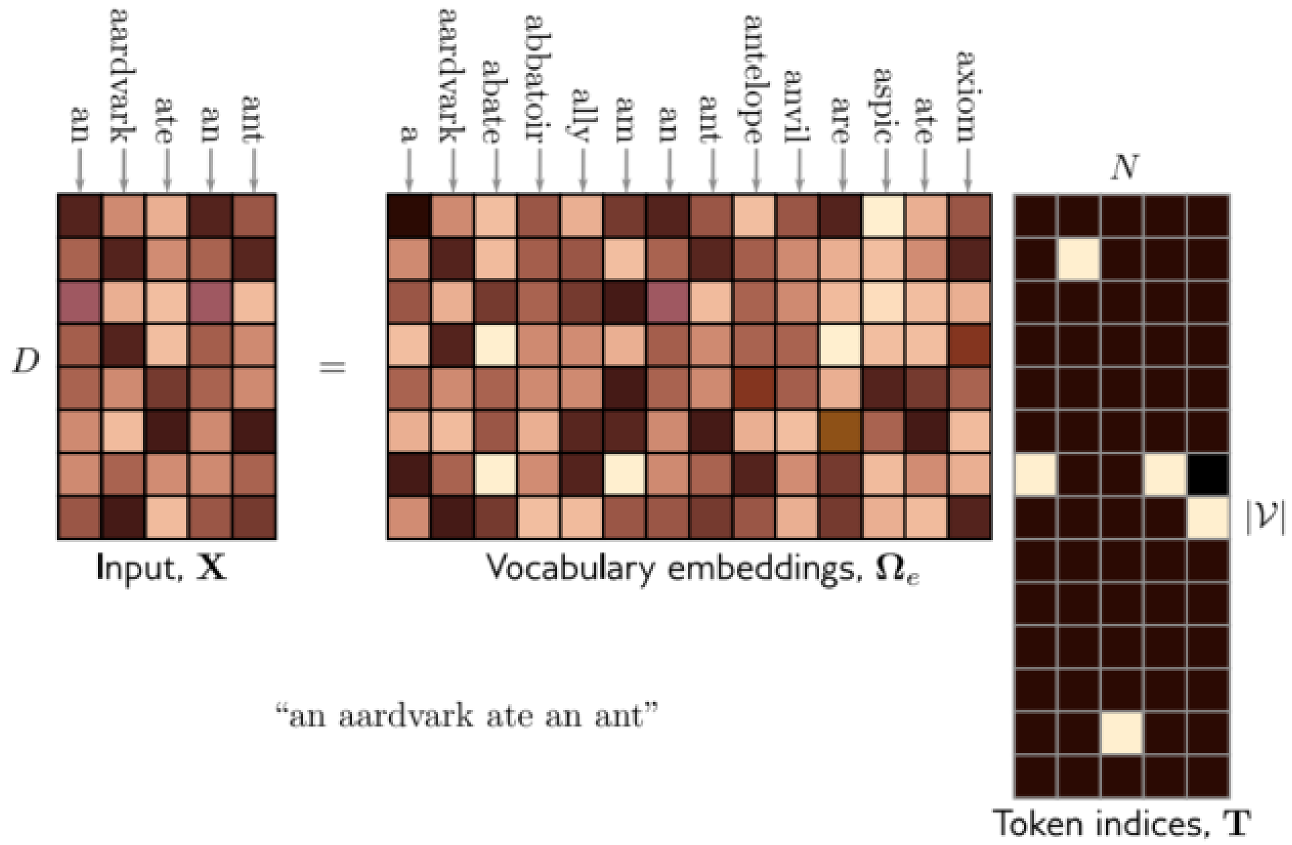



**Figure 12.9** The input embedding matrix $\mathbf{X} \in \mathbb{R}^{D\times N}$ contains $N$ embeddings of length $D$ and is created by multiplying a matrix $\boldsymbol{\Omega}_e$ containing the embeddings for the entire vocabulary with a matrix containing one-hot vectors in its columns that correspond to the word or sub-word indices. The vocabulary matrix $\boldsymbol{\Omega}_e$ is considered a parameter of the model and is learned along with the other parameters. Note that the two embeddings for the word an in $\mathbf{X}$ are the same.

# Transformer model

- Encoder

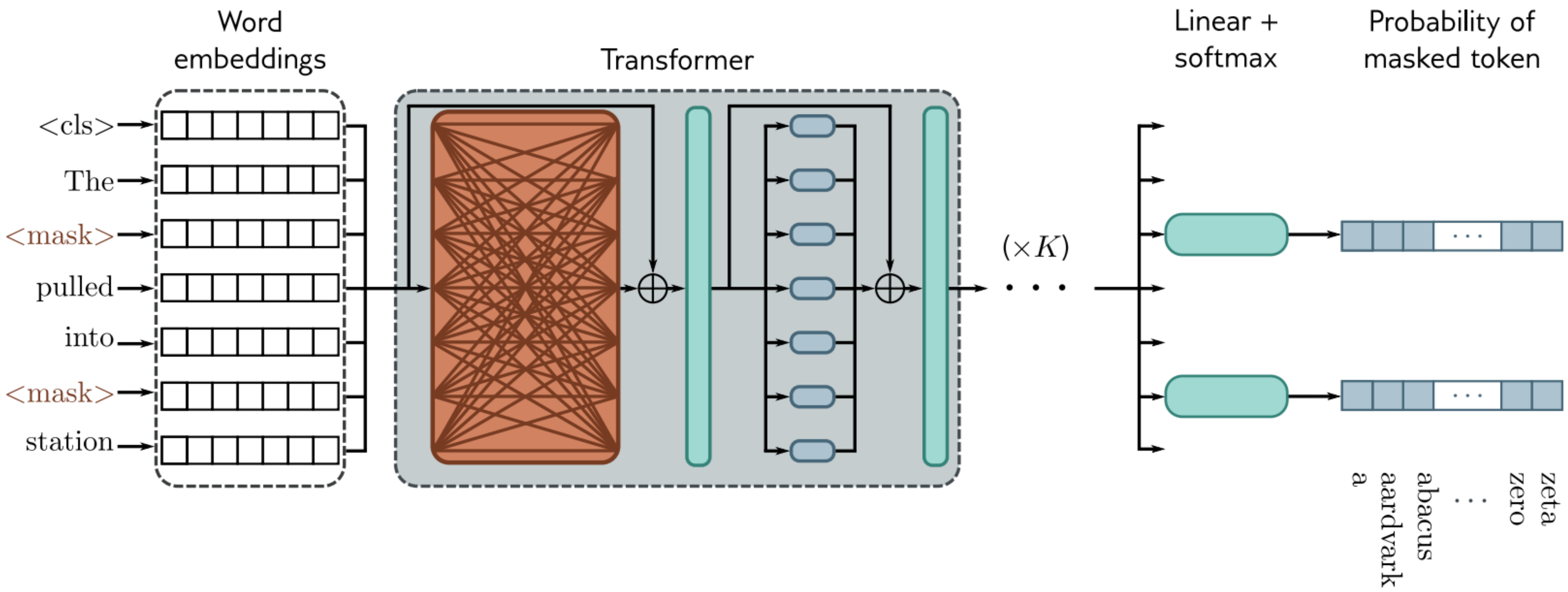
  Transforms the text embeddings into a representation that can support a variety of tasks.
- Decoder

  Predicts the next token to continue the input text.
- Encoder-Decoders:

  Used in sequence-to-sequence tasks, where one text string is converted into another (e.g., machine translation).

- Encoder model example: BERT
  - 30,000 tokens; 1024dim word embeddings
  - 24 transformers
  - 16 heads, the matrices $\mathbf{\Omega}_{vh}$, $\mathbf{\Omega}_{qh}$, $\mathbf{\Omega}_{kh}$ are $1024 \times 64$

# Pre-training

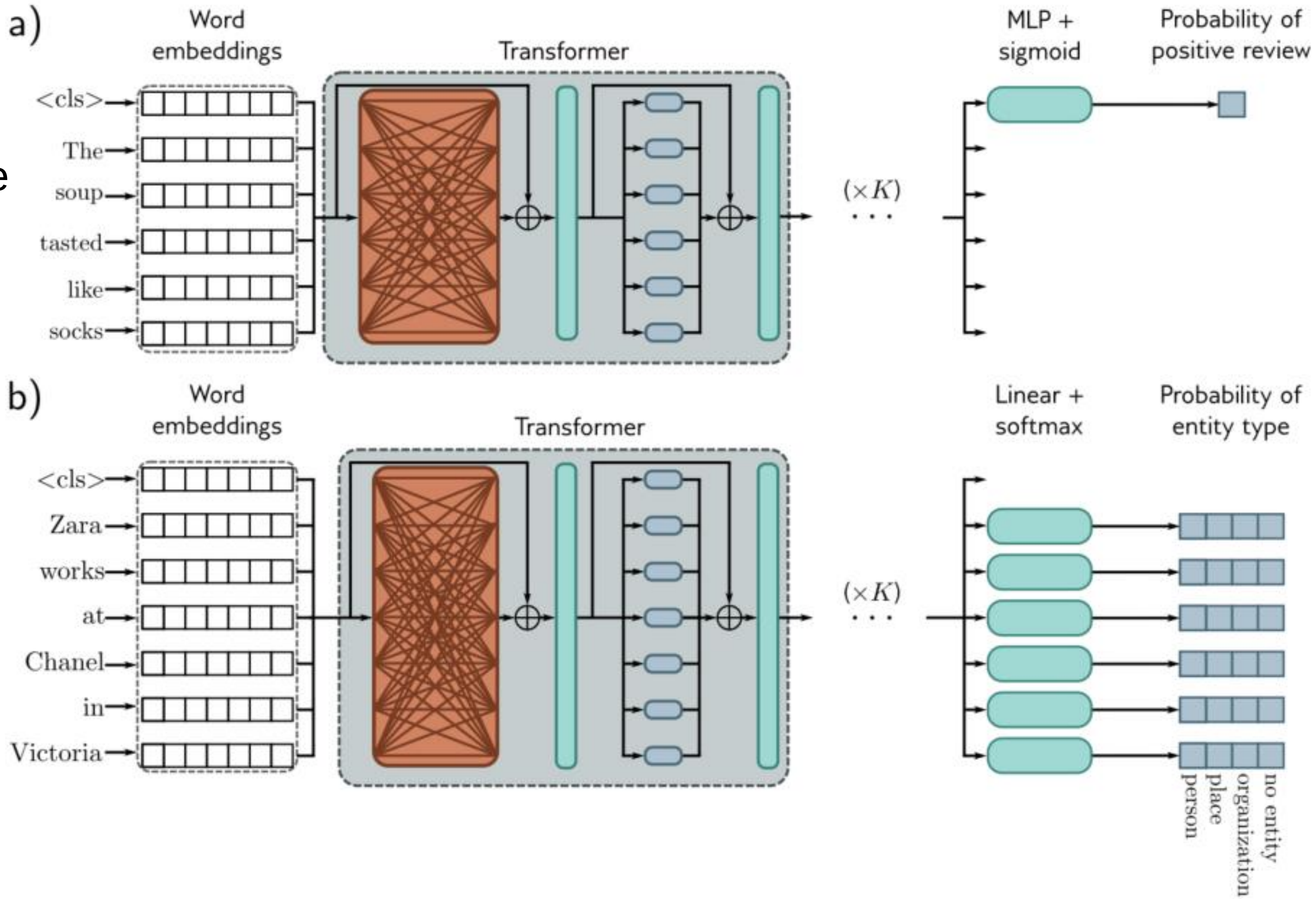



- Pre-training for BERT-like encoder. A small fraction of the input tokens is randomly replaced with a generic <mask> token.

# Fine-tuning

In the fine-tuning stage, the model parameters are adjusted to specialize the network to a particular task：

- Text classification
- Word classification



# Decoder model example: GPT3

The basic architecture is extremely similar to the encoder model and comprises a series of transformers that operate on learned word embeddings.

**Goals (Encoder and Decoder):**

- The encoder aimed to build a representation of the text that could be fine-tuned to solve a variety of more specific NLP tasks.
- The decoder has one purpose: to generate the next token in a sequence. It can generate a coherent text passage by feeding the extended sequence back into the model.

## Language modeling

Example： It takes great courage to let yourself appear weak.

# Language modeling

- The autoregressive formulation demonstrates the connection between maximizing the log probability of the tokens in the loss function and the next token prediction task.

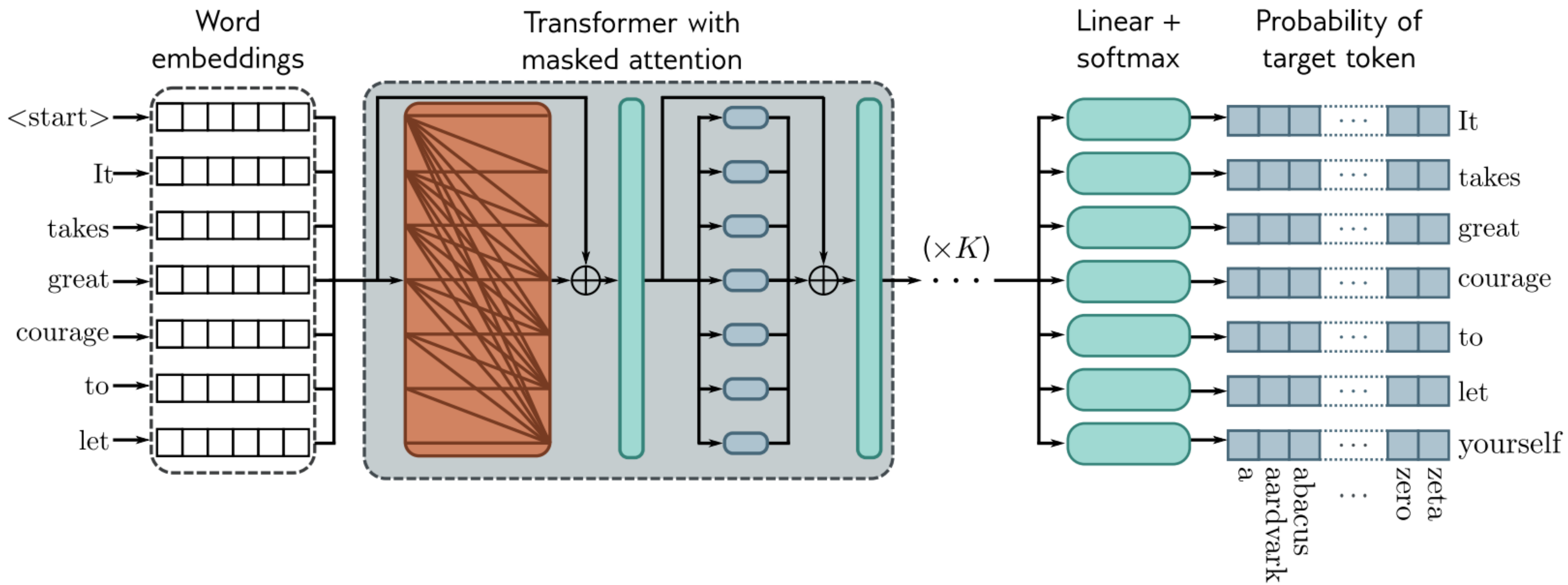
$$p(t_1, t_2, \cdots, t_N) = p(t_1)\prod_{n=1}^{N} p(t_n \mid t_1, t_2, \cdots, t_{n-1})$$

- Example： It takes great courage to let yourself appear weak.

$$\begin{aligned}
Pr(\text{It takes great courage to let yourself appear weak}) \quad = \\
& Pr(\text{It}) \times Pr(\text{takes}|\text{It}) \times Pr(\text{great}|\text{It takes}) \times Pr(\text{courage}|\text{It takes great}) \times \\
& Pr(\text{to}|\text{It takes great courage}) \times Pr(\text{let}|\text{It takes great courage to}) \times \\
& Pr(\text{yourself}|\text{It takes great courage to let}) \times \\
& Pr(\text{appear}|\text{It takes great courage to let yourself}) \times \\
& Pr(\text{weak}|\text{It takes great courage to let yourself appear}).
\end{aligned}$$

# Masked self-attention

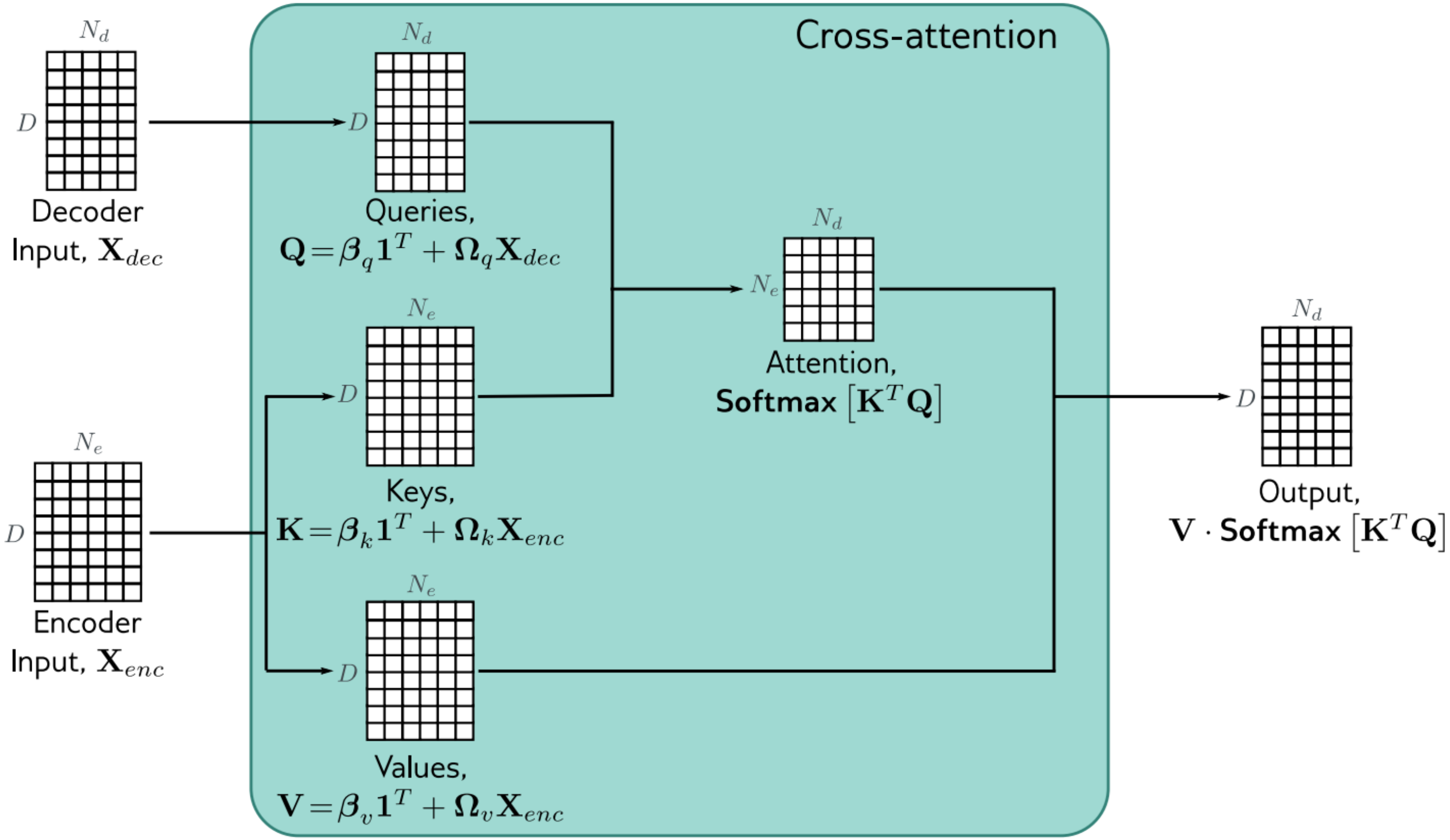
- Problem in training
  - Computing $\log[P(\text{great}|\text{It takes})]$ Accessible to “courage to let yourself appear weak”
- Solution: *masked self-attention*
  - The effect is to make the weight of all the upward-angled arrows zero



# Cross-attention

- The flow of computation is the same as in standard self-attention.
  - The queries are now calculated from the decoder embeddings $\mathbf{X}_{dec}$,
  - The keys and values from the encoder embeddings $\mathbf{X}_{enc}$

- Typical Applications
  - Machine translation
  - Multi-modal



# Transformers for images

- ImageGPT is a transformer decoder
  - an autoregressive model of image pixels that ingests a partial image and predicts the subsequent pixel value.
  - the quadratic complexity of the transformer network
  - the original 24-bit RGB color space had to be quantized into a nine-bit color space, so the system ingests (and predicts) one of 512 possible tokens at each position.
- The internal representation of this decoder was used as a basis for image classification.

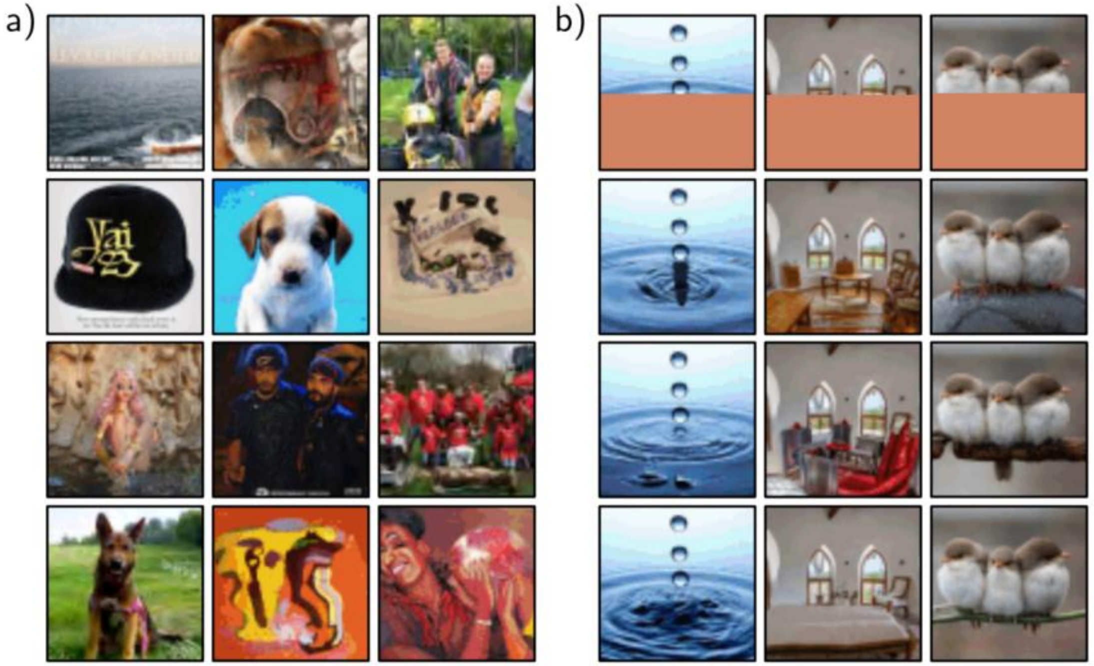
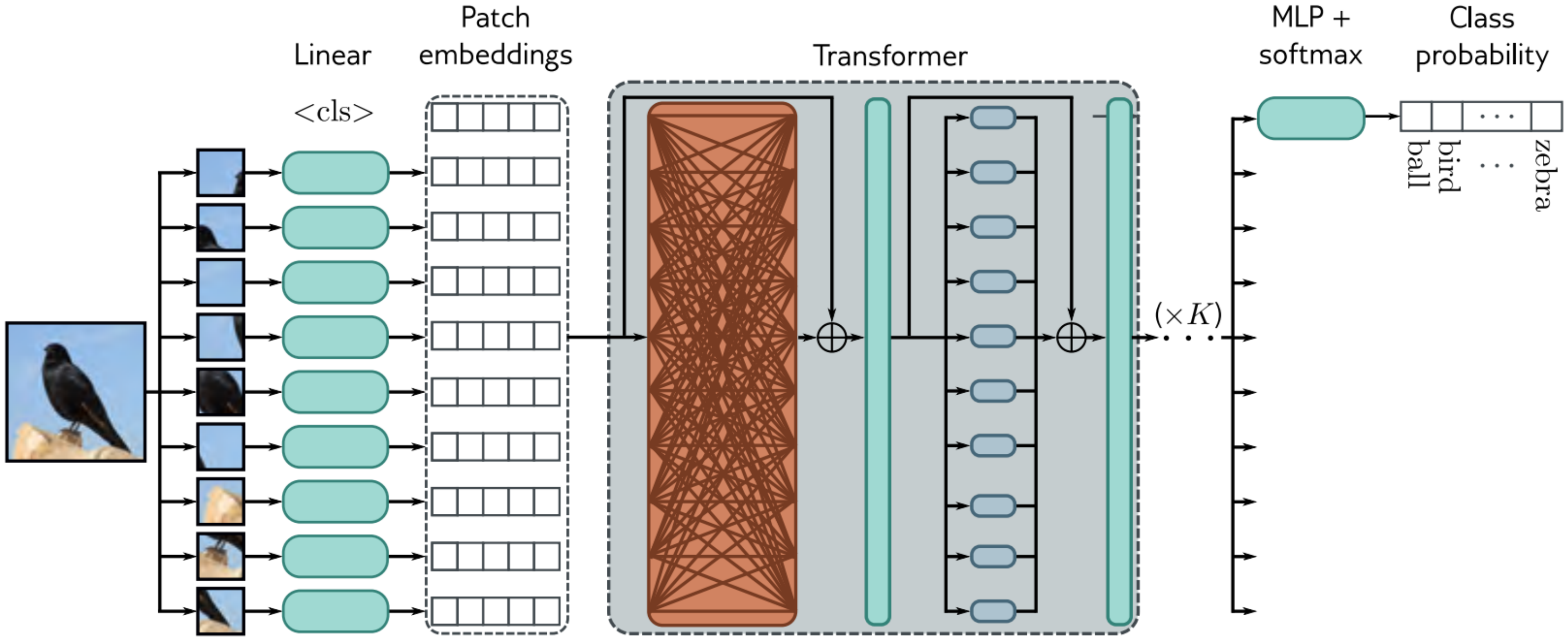



**Figure 12.16** ImageGPT. a) Images generated from the autoregressive ImageGPT

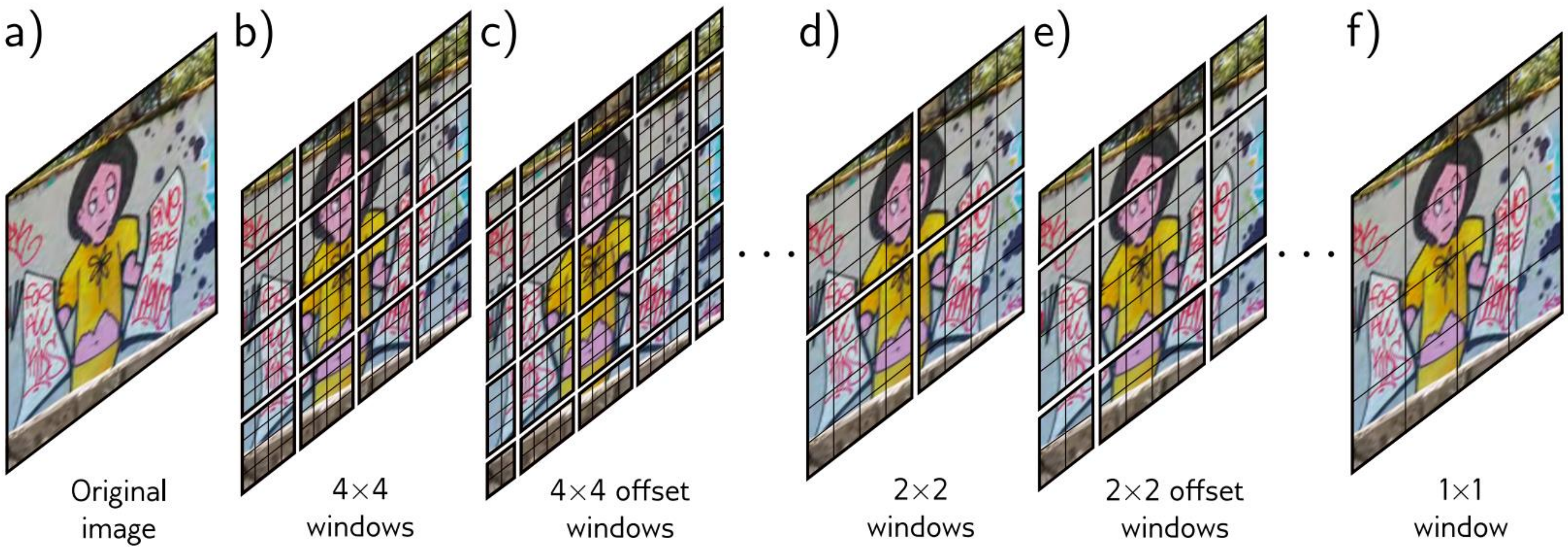
# Vision Transformer (ViT)

- The *Vision Transformer*
  - Dividing the image into 16×16 patches
  - Each patch is mapped to a lower dimension via a learned linear transformation
  - These representations are fed into the transformer network. Once again, standard 1D positional encodings are learned.



# Multi-scale vision transformers

- The Vision Transformer
  - Several transformer models that process the image at multiple scales
  - Start with high-resolution patches and few channels
  - Gradually decrease the resolution, while increasing the number of channels.



# Summary

- Self-attention and the transformer architecture
- The transformer operates on sets of high-dimensional embeddings
- Describe long-range dependencies in text
- Decoders build an autoregressive model