IERG5050 AI Foundation Models, Systems and Applications
Spring 2025

Transformers Part I: Basic Architecture

Prof. Wing C. Lau
wclau@ie.cuhk.edu.hk
http://www.ie.cuhk.edu.hk/wclau

Acknowledgements

Many of the slides in this lecture are adapted from the sources below. Copyrights belong to the original authors.

- Stanford CS25: Transformer United V4, Spring 2024, https://web.stanford.edu/class/cs25/
  Instructors: Div Garg, Steven Feng, Seonghee Lee, Emily Bunnapradist ;
  Faculty Advisor: Prof. Chris Manning,
  Overview Slides https://docs.google.com/presentation/d/1oXPs3LXtIVIsVbwTyGjAWj_aWvak9c1uNC4uhkS6glk/edit?usp=sharing
- Stanford CS336: Language Modeling from Scratch, Spring 2024
  - by Profs. Tatsunori Hashimoto, Percy Liang, https://stanford-cs336.github.io/spring2024/
- Stanford CS229S: Systems for Machine Learning, Fall 2023
  by Profs. Azalia Mirhoseini, Simran Arora, https://cs229s.stanford.edu/fall2023/
- Stanford CS224N: Natural Language Processing with Deep Learning, Winter 2021
  by Prof. Chris Manning, https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/
- Stanford CS231n: Deep Learning for Computer Vision, Spring 2023
  by Prof. Fei-fei Li, https://cs231n.stanford.edu/slides/2023/
- CMU 11-667: Large Language Models: Methods and Applications, Fall 2024
  by Profs. Chenyan Xiong and Daphne Ippolito, https://cmu-llms.org
- CMU 11-711: Advanced Natural Language Processing (ANLP), Spring 2024
  by Prof. Graham Neubig, https://phontron.com/class/anlp2024/lectures/
- UPenn CIS7000: Large Language Models, Fall 2024
  by Prof. Mayur Naik, https://llm-class.github.io/schedule.html
- Princeton COS597G: Understanding Large Language Models, Fall 2022
  by Prof. Danqi Chen, https://www.cs.princeton.edu/courses/archive/fall22/cos597G/
- UWaterloo CS886: Recent Advances on Foundation Models, Winter 2024
  by Prof. Wenhu Chen, https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/
- UMD CMSC848K: Multimodal Foundation Models, Fall 2024
  by Prof. Jia-Bin Huang, https://jbhuang0604.github.io/teaching/CMSC848K/

Natural Language Processing (NLP) & Language Modeling

- NLP (natural language processing) tasks
  - Translation, question answering, recommendations, sentence completion, etc
- Language model
  - Model the probability of a sequence of tokens in a text
- Examples
  - I was eating __.
    - an apple (0.02)
    - a banana (0.01)
    - popcorns (0.0001)
  - I was in a __. I was eating popcorns
    - house (0.01)
    - mansion (0.001)
    - movie theater (0.2)

$$p(x) = p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | x_{<t})$$

Representing Words as Discrete Symbols

In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel – a localist representation

Means one 1, the rest 0s

Such symbols for words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

Problem with Words as Discrete Symbols

Example: in web search, if a user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"

But:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal

There is no natural notion of **similarity** for one-hot vectors!

Solution:

- Could try to rely on WordNet's list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

Representing Words by their Context

- Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**
  - *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of *w* to build up a representation of *w*

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent ***banking***

Word Vectors (aka Word Embeddings)

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector dot (scalar) product

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \qquad \textit{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}$$

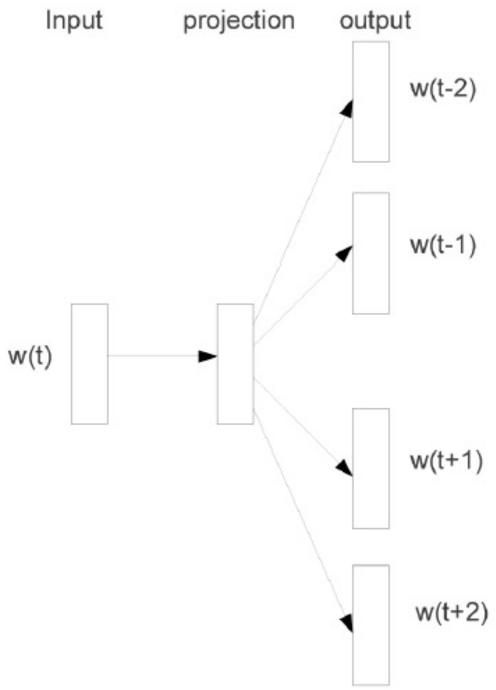
Note: word vectors are also called (word) embeddings or (neural) word representations
They are a distributed representation

Word2vec: How to learn the Word Embedding

Word2vec is a framework for learning word vectors (Mikolov et al. 2013)

Idea:

- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
- Keep adjusting the word vectors to maximize this probability

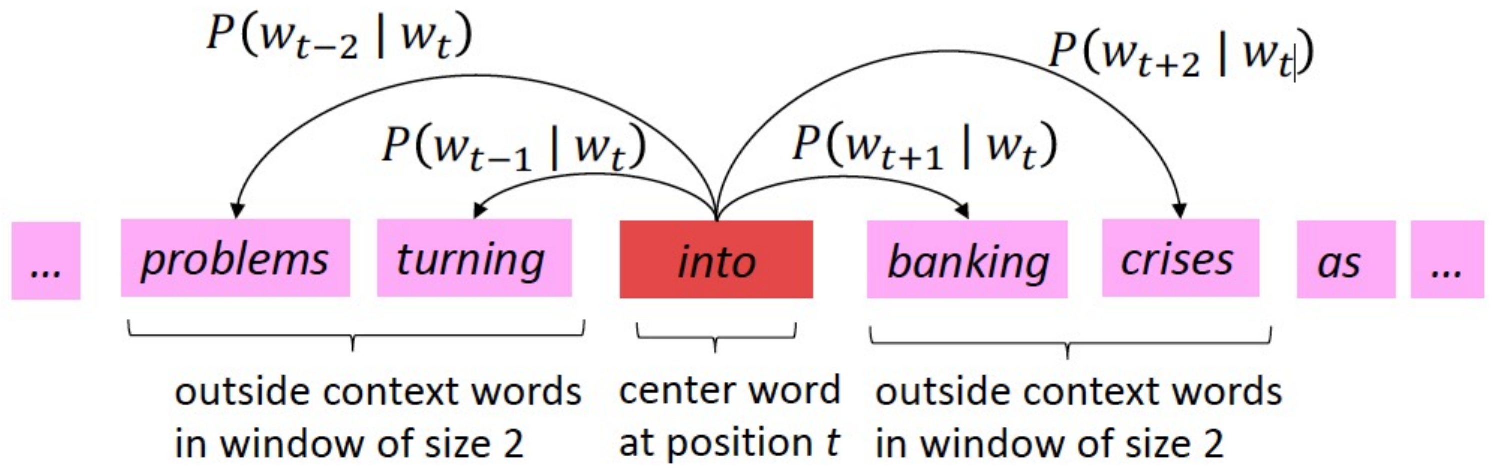


Skip-gram model (Mikolov et al. 2013)

Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/slides/cs224n-2024-lecture01-wordvecs1-public.pdf
For details, refer to: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/readings/cs224n_winter2023_lecture1_notes_draft.pdf

Word2vec Overview

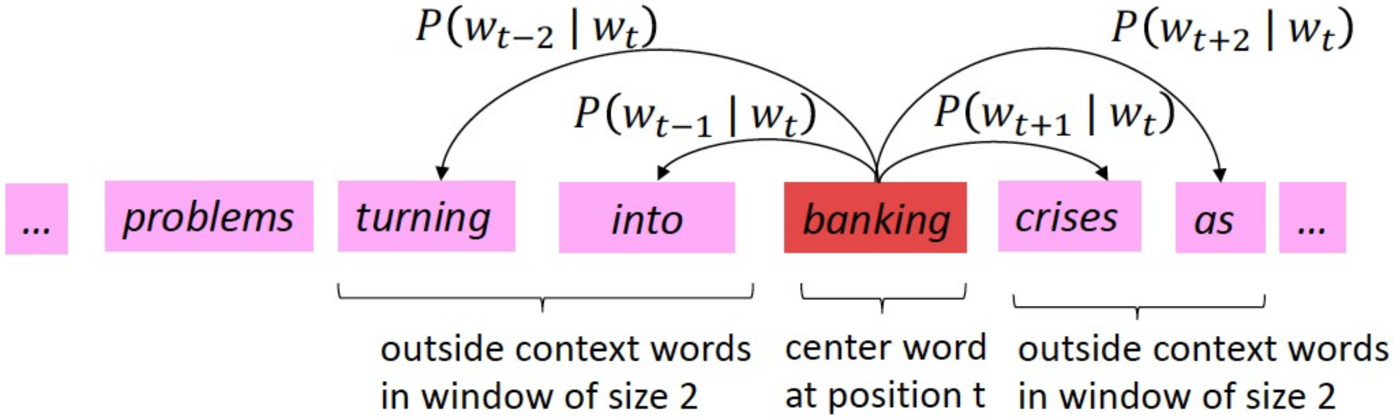
Example windows and process for computing $P(w_{t+j} \mid w_t)$



Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/slides/cs224n-2024-lecture01-wordvecs1-public.pdf
For details, refer to: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/readings/cs224n_winter2023_lecture1_notes_draft.pdf

Word2vec Overview

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/slides/cs224n-2024-lecture01-wordvecs1-public.pdf
For details, refer to: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/readings/cs224n_winter2023_lecture1_notes_draft.pdf

Word2vec: Objective Function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_t. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/slides/cs224n-2024-lecture01-wordvecs1-public.pdf
For details, refer to: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/readings/cs224n_winter2023_lecture1_notes_draft.pdf

Word2vec: Objective Function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\ne 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

- **Question:** How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?
- **Answer:** We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word

  } These word vectors are subparts of the big vector of all parameters θ
- Then for a center word *c* and a context word *o*:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/slides/cs224n-2024-lecture01-wordvecs1-public.pdf
For details, refer to: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/readings/cs224n_winter2023_lecture1_notes_draft.pdf

Word2vec: Prediction Function = Prob[o|c]

② Exponentiation makes anything positive

① Dot product compares similarity of o and c.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

$$P(o|c) = \frac{\exp(\boxed{u_o^T v_c})}{\boxed{\sum_{w \in V} \exp(u_w^T v_c)}}$$

③ Normalize over entire vocabulary
to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \rightarrow (0,1)^n$

Open
region

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

But sort of a weird name
because it returns a distribution!

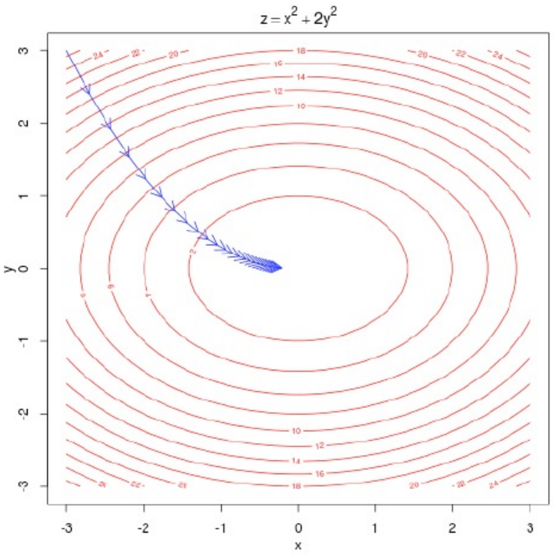
Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/slides/cs224n-2024-lecture01-wordvecs1-public.pdf
For details, refer to: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/readings/cs224n_winter2023_lecture1_notes_draft.pdf

Word2vec: To Train the Model

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with d-dimensional vectors and V-many words, we have →
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute all vector gradients!

Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/slides/cs224n-2024-lecture01-wordvecs1-public.pdf
For details, refer to: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/readings/cs224n_winter2023_lecture1_notes_draft.pdf

The problematic P[o|c]

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{i=1}^{N} \exp(u_i^T v_c)}$$

The denominator is difficult to evaluate as
it involves the embedding of ALL words in the Universe (Vocabulary)

=> Use a trick to circumvent the problem via Negative Sampling

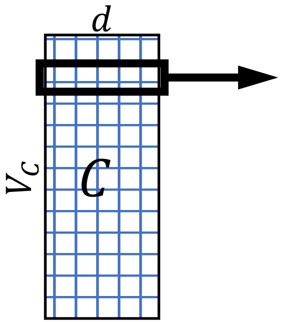
What is Word2vec ?

- `word2vec` is **not** a single algorithm
- It is a **software package** for representing words as vectors, containing:
  - Two distinct models
    - CBoW
    - **Skip-Gram** **(SG)**
  - Various training methods
    - **Negative Sampling** **(NS)**
    - Hierarchical Softmax
  - A rich preprocessing pipeline
    - Dynamic Context Windows
    - Subsampling
    - Deleting Rare Words
- We will focus on the Skip-Grams with Negative Sampling (SGNS) approach !

Slide by Omer Levy

SGNS starts with SAME basic setting

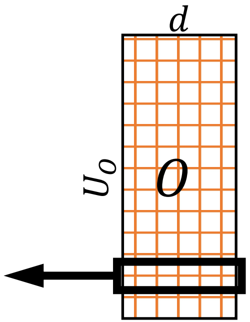
- SGNS finds a vector v_c for each word c in our vocabulary V_C
- Each such vector has d latent dimensions (e.g. $d = 100$)
- Effectively, it learns a matrix C whose rows represent the center vectors v_c
- **Key point:** it also learns a similar auxiliary matrix O of <u>outside context vectors</u>
- In fact, each word has two embeddings: v_c and u_o



c:wampimuk =
$(-3.1, 4.15, 9.2, -6.5, \dots)$

$\neq$

o:wampimuk =
$(-5.6, 2.95, 1.4, -1.3, \dots)$



“word2vec Explained…”
Goldberg & Levy, arXiv
2014

Skip-Grams with Negative Sampling (SGNS)

You first observe (actually sample) the following sentence from the training corpus:

Marco saw a furry little wampimuk hiding in the tree.

"word2vec Explained…"
Goldberg & Levy, arXiv 2014

Skip-Grams with Negative Sampling (SGNS)

Marco saw a furry little wampimuk hiding in the tree.

center word	**outside context word**
wampimuk	furry
wampimuk	little
wampimuk	hiding
wampimuk	in
…	…

D (observed data)

"word2vec Explained…" Goldberg & Levy, arXiv 2014

Skip-Grams with Negative Sampling (SGNS)

Maximize $\prod_i \sigma(\vec{c} \cdot \overrightarrow{o_i})$

- o_i was **observed** with c

where $\sigma(z) = 1/\ [1+\exp(-z)]$

<u>**center word**</u>	<u>**outside context**</u>
wampimuk	furry
wampimuk	little
wampimuk	hiding
wampimuk	in

AND

Minimize $\prod_i \sigma(\vec{c} \cdot \overrightarrow{o_i}')$

≡ **Maximize** $\prod_i [1 - \sigma(\vec{c} \cdot \overrightarrow{o_i}')]$

- o_i' was NOT **observed** with c, they are from the set of Negative Samples *D'* randomly generated by the algorithm.

<u>**center word**</u>	<u>**NOT outside context**</u>
wampimuk	Australia
wampimuk	cyber
wampimuk	the
wampimuk	1985

Take Log and the optimization problem becomes "similar" to the training of a binary logistic-regression classifier:

$$\arg\max_\theta \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c \cdot v_w}} + \sum_{(w,c)\in D'} \log(1 - \frac{1}{1+e^{-v_c \cdot v_w}}) = \arg\max_\theta \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c \cdot v_w}} + \sum_{(w,c)\in D'} \log(\frac{1}{1+e^{v_c \cdot v_w}})$$

Summary: How to learn Word2vec Embeddings via SGNS

For a vocabulary of size V: Start with V random 300-dimensional vectors as initial embeddings

Train a logistic regression classifier to distinguish words that co-occur in corpus from those that don't

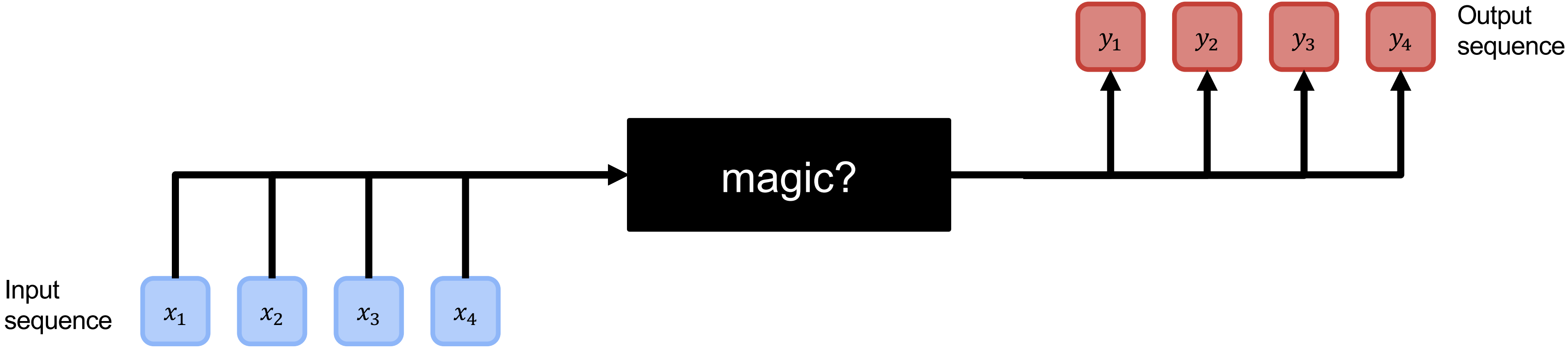
- Pairs of words that co-occur are positive examples
- Pairs of words that don't co-occur are negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance

Throw away the classifier code and keep the embeddings.

NLP Tasks as Sequence to Sequence Modeling

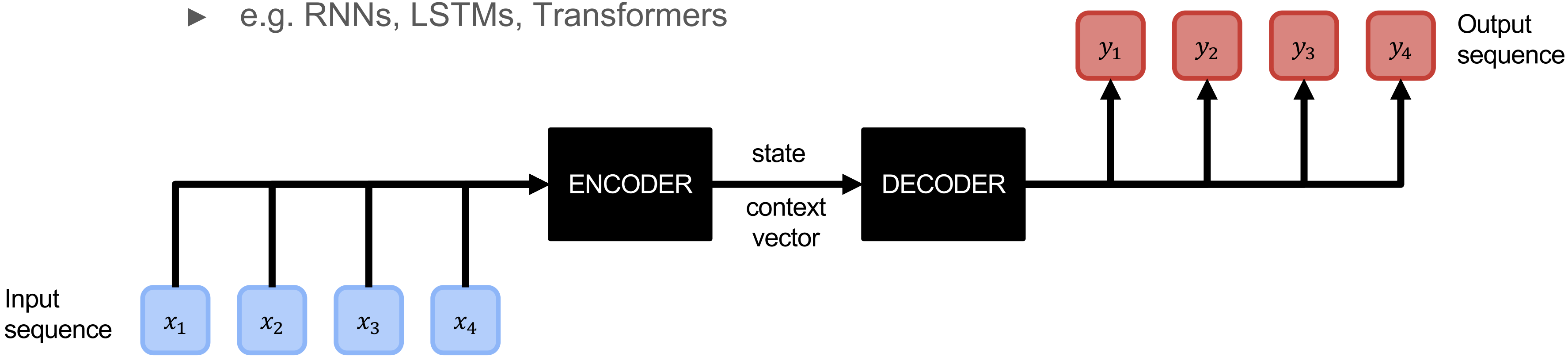
NLP Tasks as Sequence to Sequence Modeling

- **Example Scenarios**
  - Text → Text (e.g. Q/A, translation, text summarization)
  - Image → Text (e.g. image captioning)

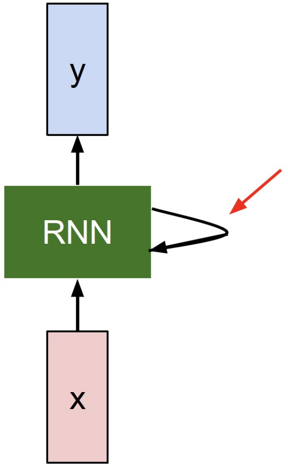


NLP Tasks as Sequence to Sequence Modeling

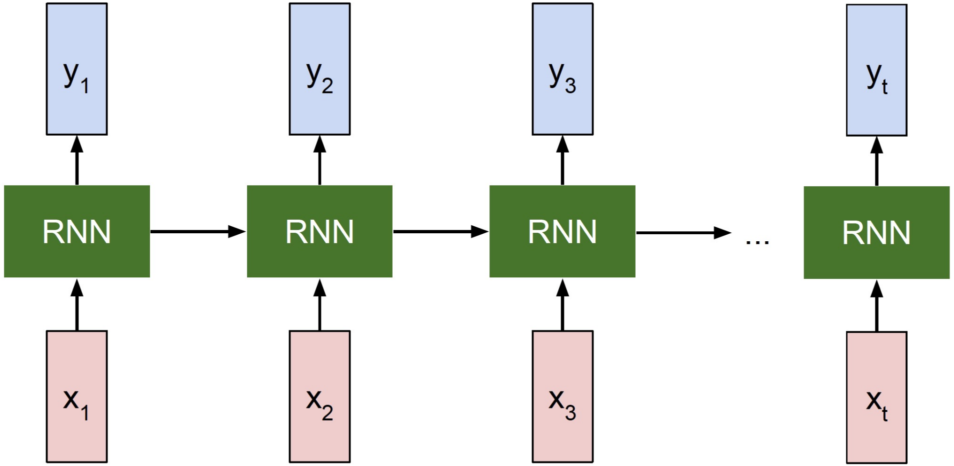
- **Example Scenarios**
  - Text → Text (e.g. Q/A, translation, text summarization)
  - Image → Text (e.g. image captioning)
- **How?** Usually Encoder-Decoder models
  - e.g. RNNs, LSTMs, Transformers



Recurrent Neural Networks (RNN) – a Seq2Seq NN model



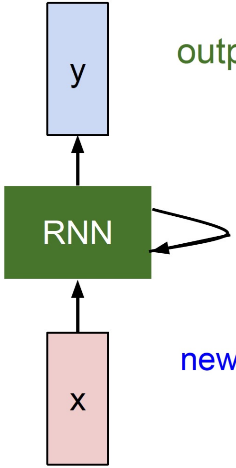
Key idea: RNNs have an "internal state" that is updated as a sequence is processed



An RNN "Unrolled" along the Time axis

Recurrent Neural Networks (RNN)

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:



$$\boxed{y_t} = \boxed{f_{W_{hy}}}(\boxed{h_t})$$

output

new state

another function with parameters W_o

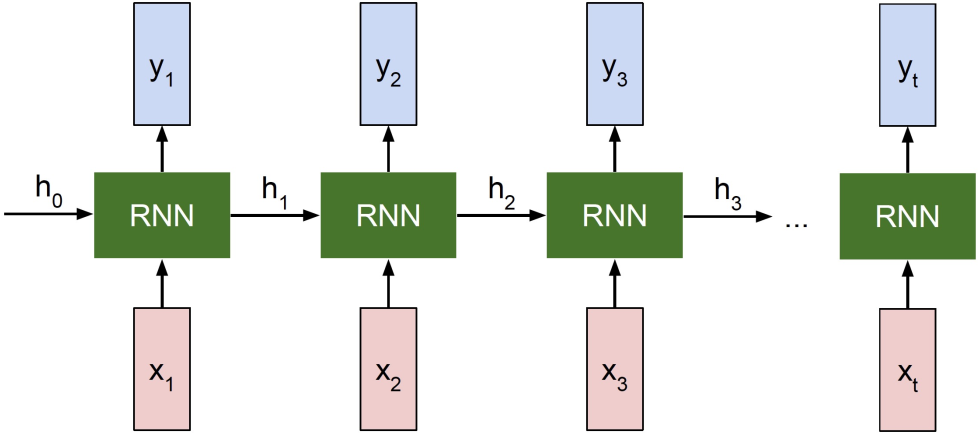
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

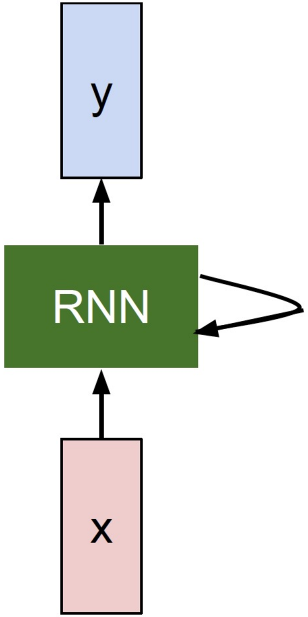
some function with parameters W



Notice: the same function and the same set of parameters are used at every time step.

(Vanilla) Recurrent Neural Networks (RNN)

The state consists of a single *"hidden"* vector **h**:



$$h_t = f_W(h_{t-1}, x_t)$$

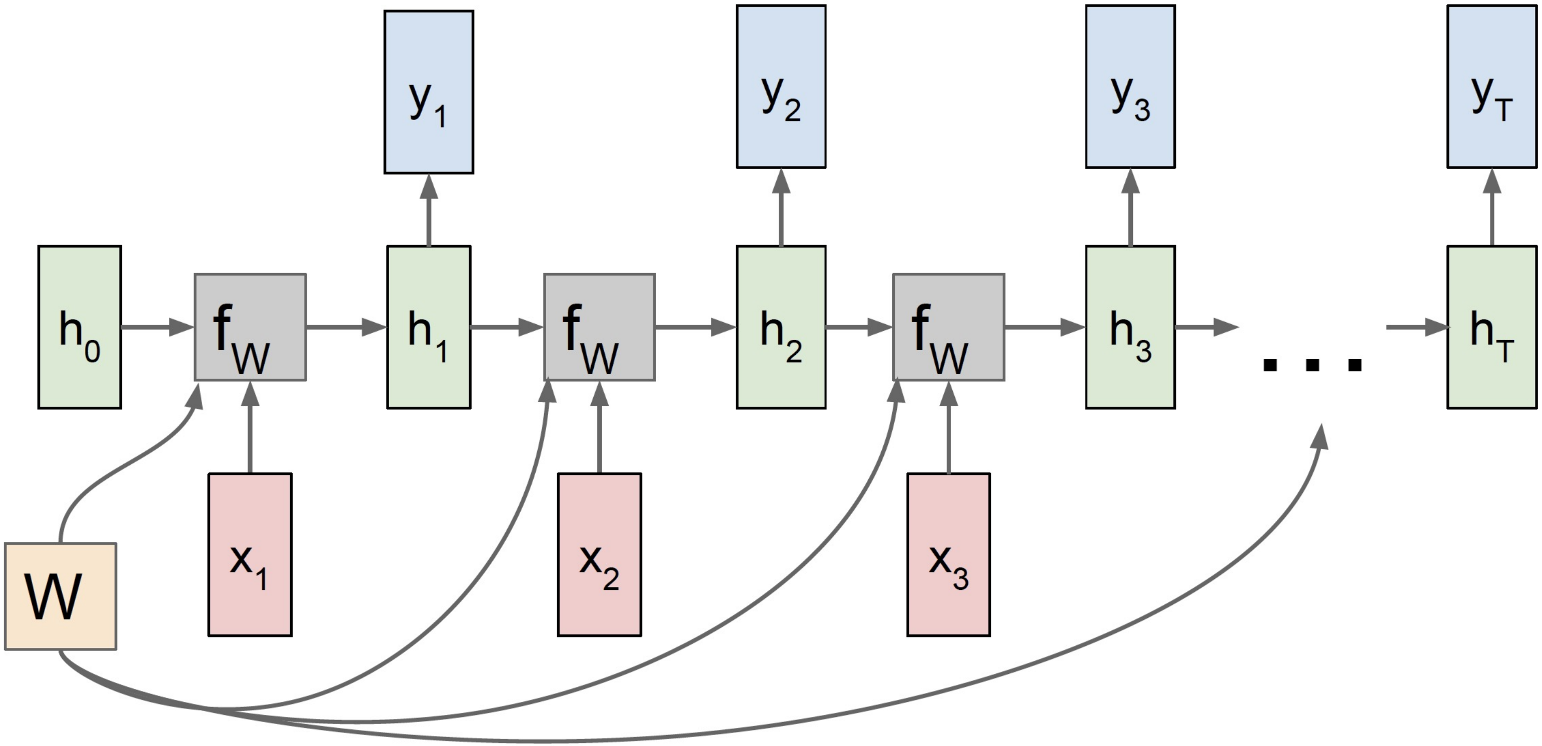
$$\downarrow$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

Computational Graph for an RNN



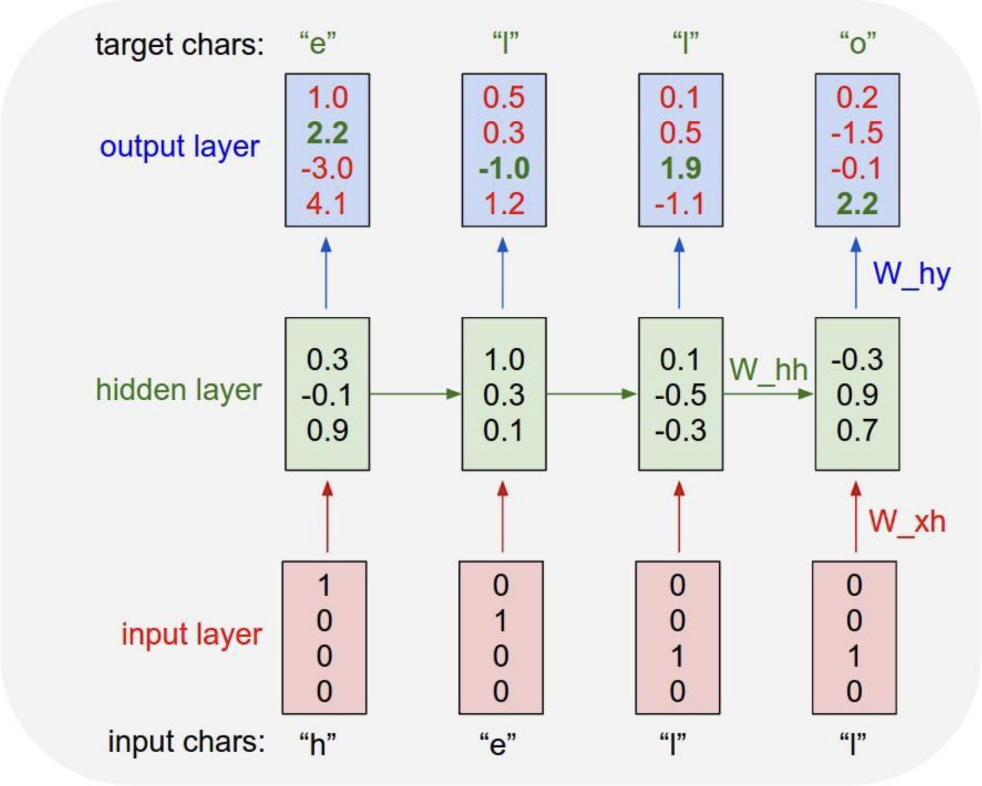
Note the reusing of the SAME weight matrix f_W at every time-step !

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

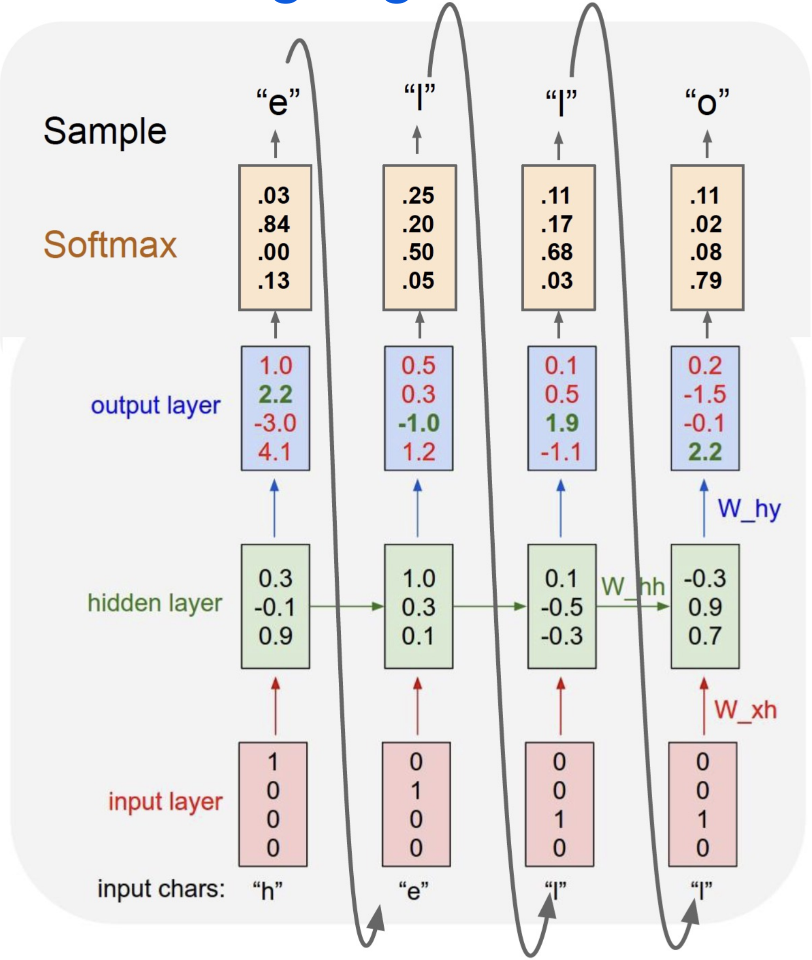
$$\boxed{h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)}$$



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

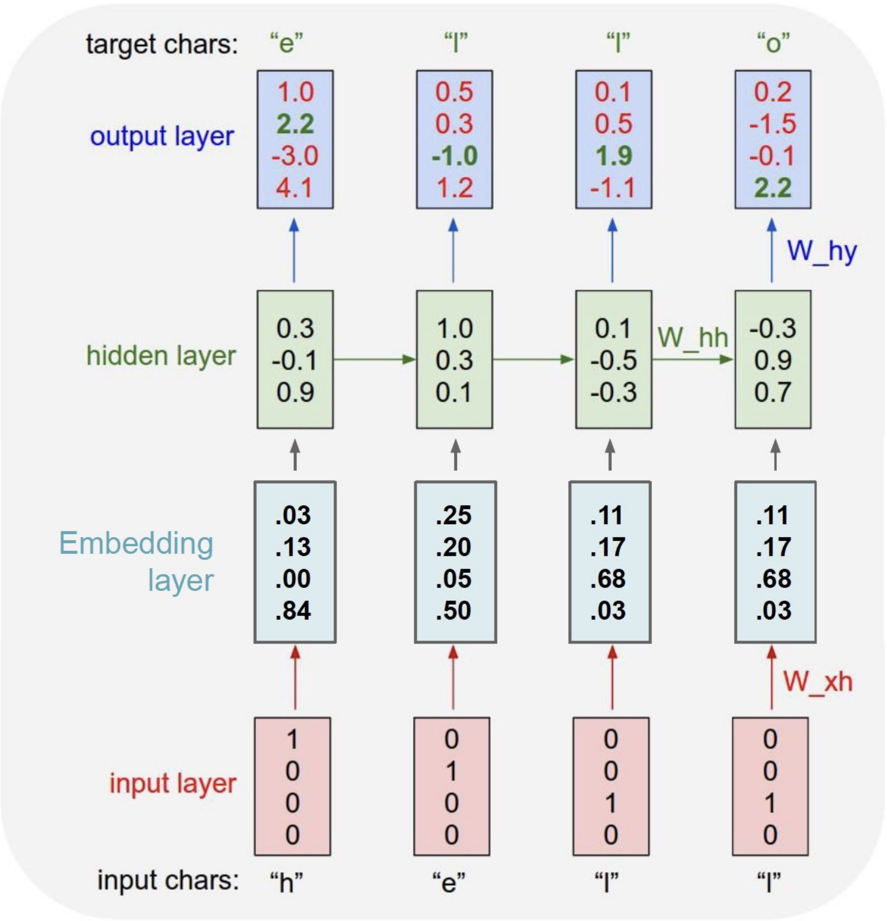
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. We often put a separate **embedding** layer between input and hidden layers.

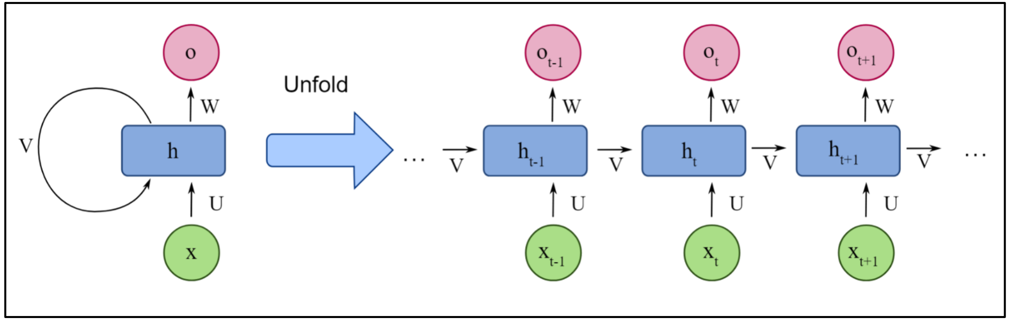


Weaknesses of early NN-based NLP approaches

- Short context length
- “Linear” reasoning - no attention mechanism to focus on other parts
- Earlier approaches (e.g. word2vec) do not adapt based on context.

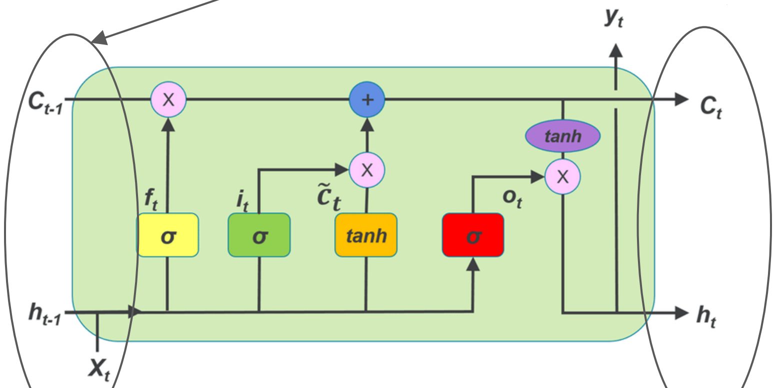
Seq2Seq Models w/ Neural Nets: the Pre-Transformer Era

- Recurrent Neural Networks (RNNs)
- Long Short-Term Memory Networks (LSTMs)
- Capture dependencies between input tokens
- Gates control the flow of information



A simple RNN shown unrolled in time. Network layers are recalculated for each time step, while weights U, V and W are shared across all time steps.

The inputs to each unit consists of the current input x_t, previous hidden state h_{t-1}, and previous context c_{t-1}

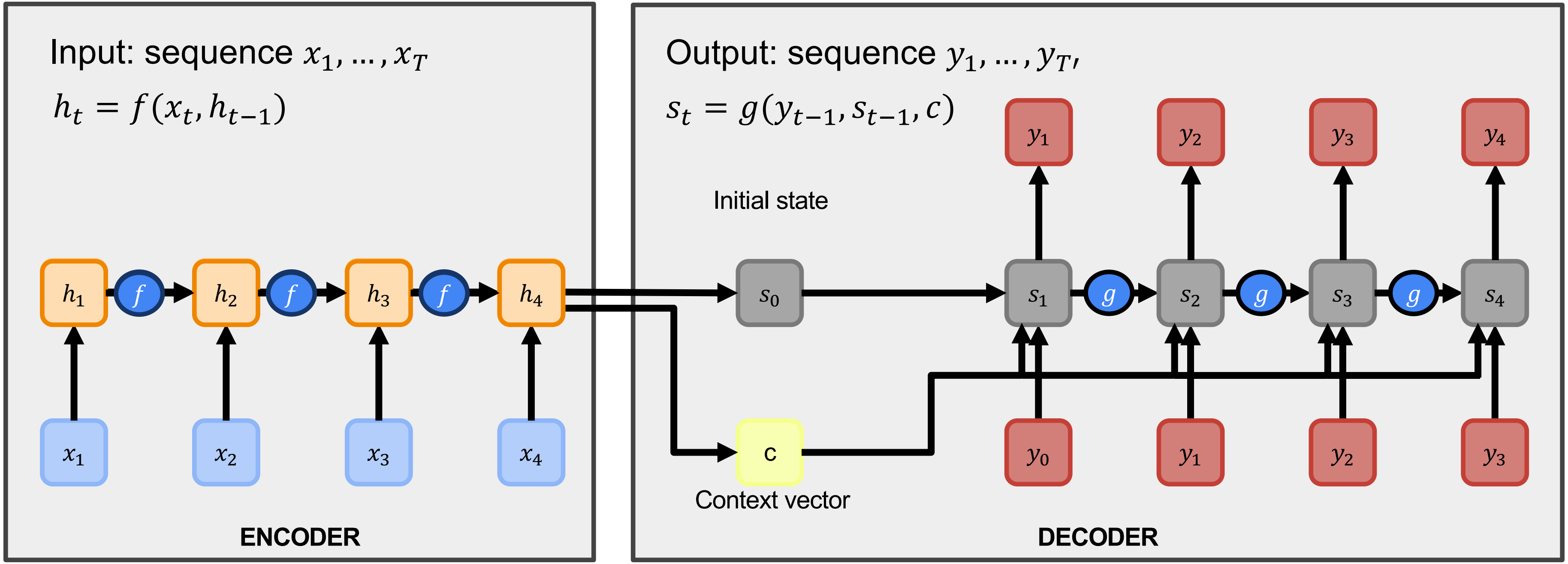


A single LSTM unit displayed as a computation graph.

The outputs are a new hidden state h_t and an updated context c_t.

Better Capturing of Long-Range Dependence using LSTM for Seq2Seq Modeling

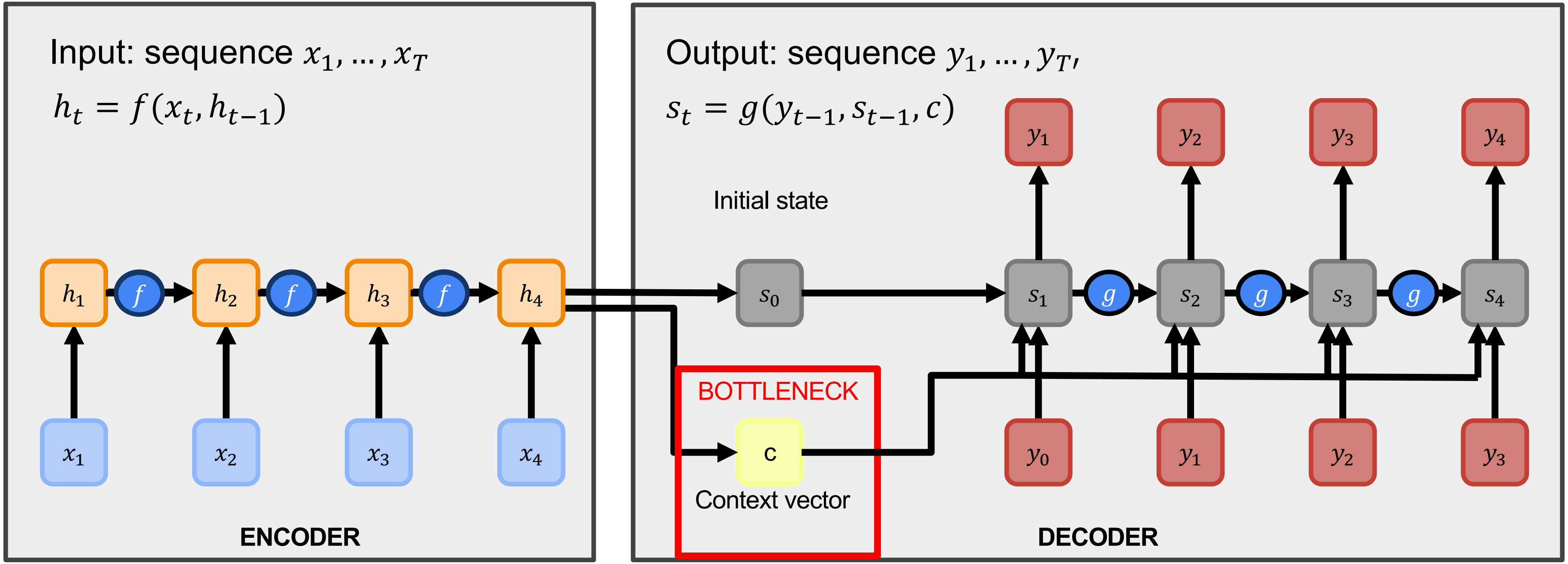
- Encoder (LSTM) and decoder (LSTM)
- Fixed-length context vector



I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS)*, 2014, pp. 3104–3112.

LSTM still suffers from Information Bottleneck

- Encoder (LSTM) and decoder (LSTM)
- Fixed-length context vector (bottleneck)



I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS)*, 2014, pp. 3104–3112.

Attention Timeline

1990s — **Prehistoric Era**

Rule-based methods, parsing, RNNs, LSTMs

2014 — **Simple attention mechanisms**

2017 — **Beginning of transformers**

Attention is all you need

2018 — **Explosion of transformers in NLP**

BERT, GPT-3

2018-2020 — **Explosion into other fields**

Explosion into other fields: ViTs, Alphafold-2.

2021-2022 — **Start of Generative Era**

Codex, Decision Transformers, GPT-X, DALL·E

2024 — **Present Day**

Huge models, more applications: Chat-GPT, GPT-4, Gemini, Llama and open-source LLMs, Whisper, Robotics Transformer, Stable Diffusion, Sora, LLM Agents, Multimodal, and so much more...!

Future (?!)

Sequence to Sequence with RNNs + **Attention**

- **Idea!** Use a different context vector for each timestep in the decoder

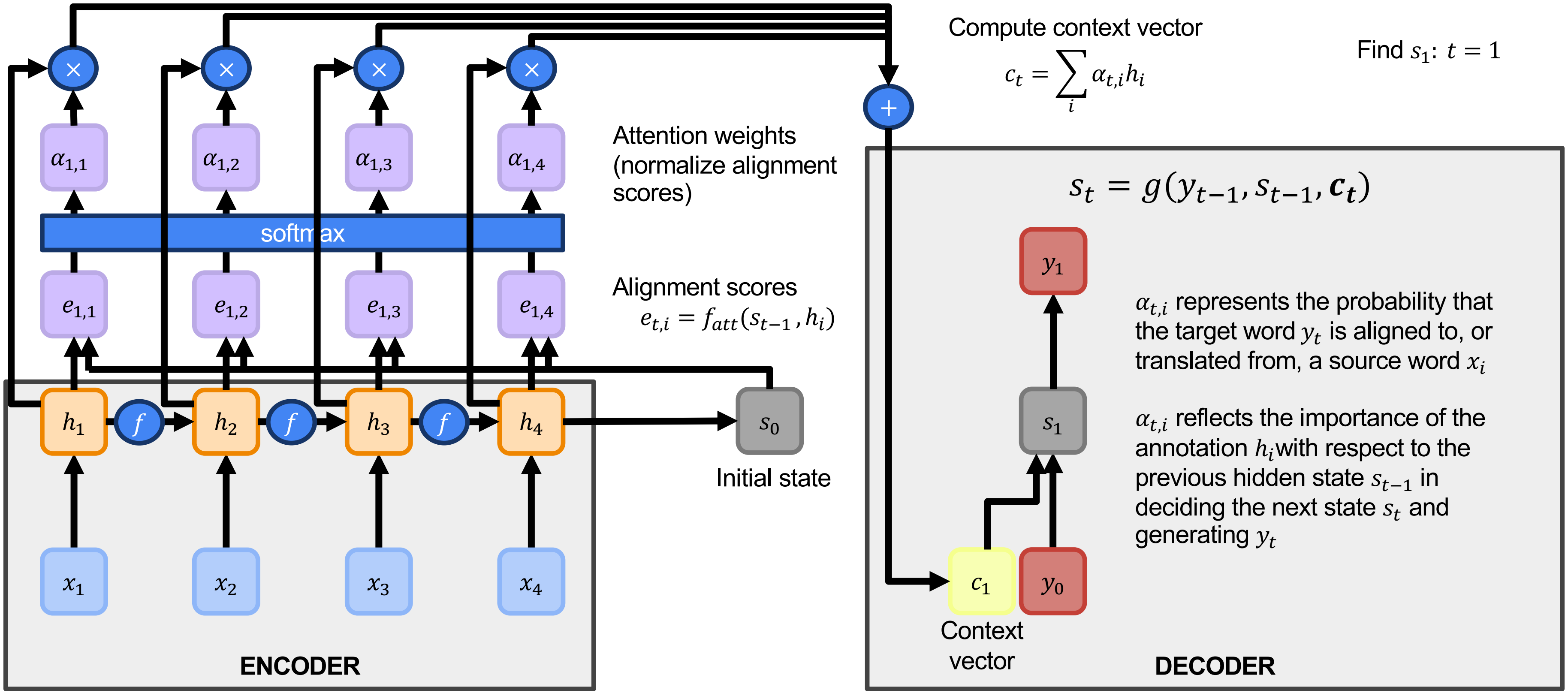
$$s_t = g(y_{t-1}, s_{t-1}, \boldsymbol{c_t})$$

- No more bottleneck through a single vector

- Craft the context vector so that it “looks at” different parts of the input sequence for each decoder timestep

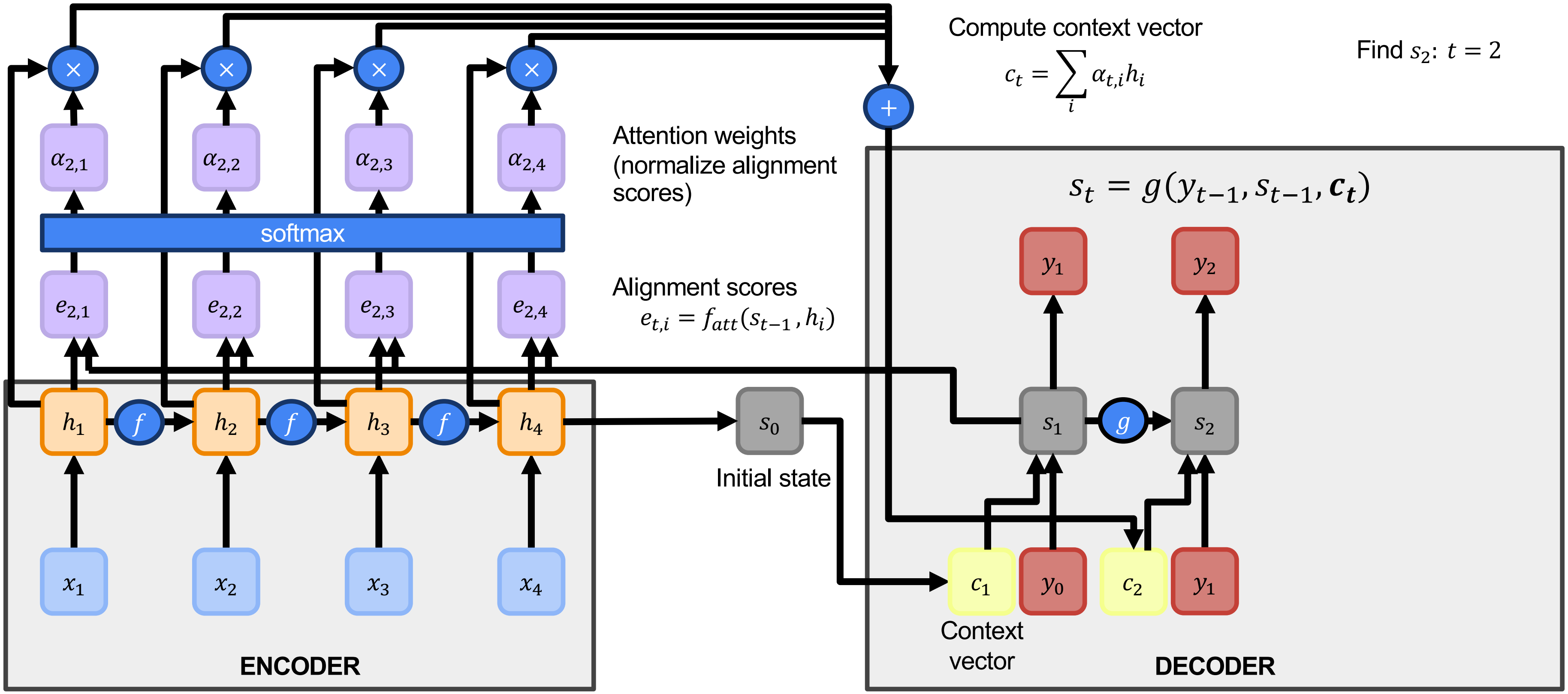
D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” in *3rd International Conference on Learning Representations (ICLR),* 2015.

Sequence to Sequence with RNNs + **Attention**



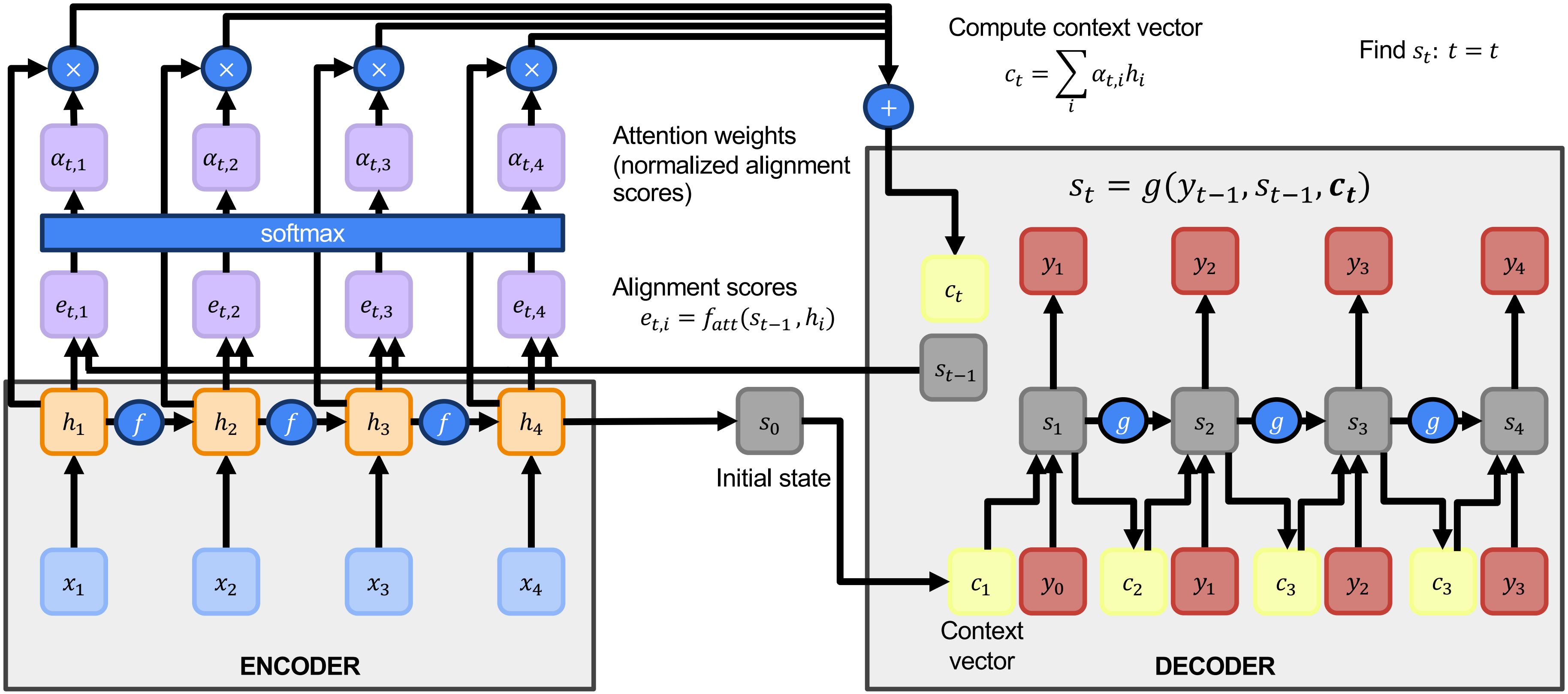
D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *3rd International Conference on Learning Representations (ICLR)*, 2015.

Sequence to Sequence with RNNs + **Attention**



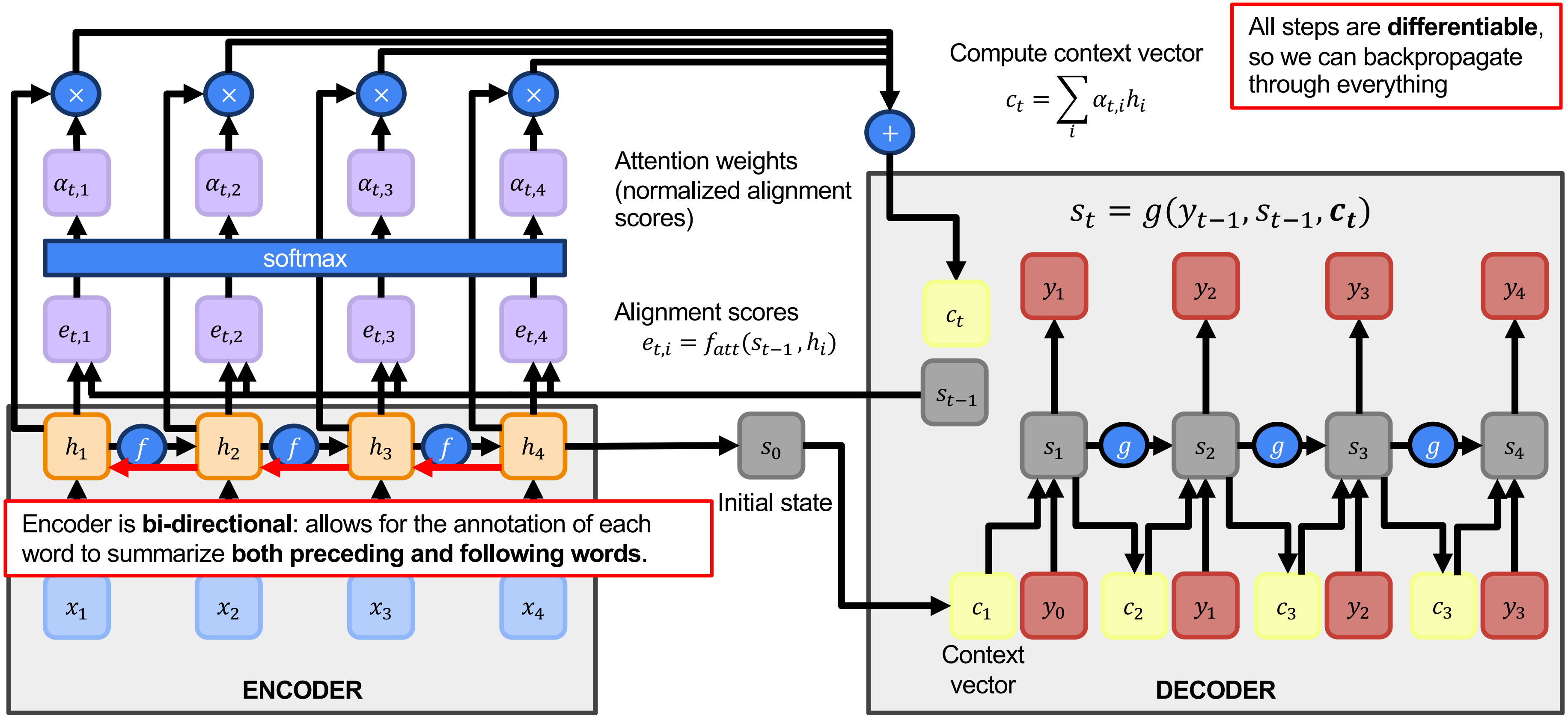
D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *3rd International Conference on Learning Representations (ICLR)*, 2015.

Sequence to Sequence with RNNs + **Attention**



D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” in *3rd International Conference on Learning Representations (ICLR)*, 2015.

Sequence to Sequence with RNNs + **Attention**

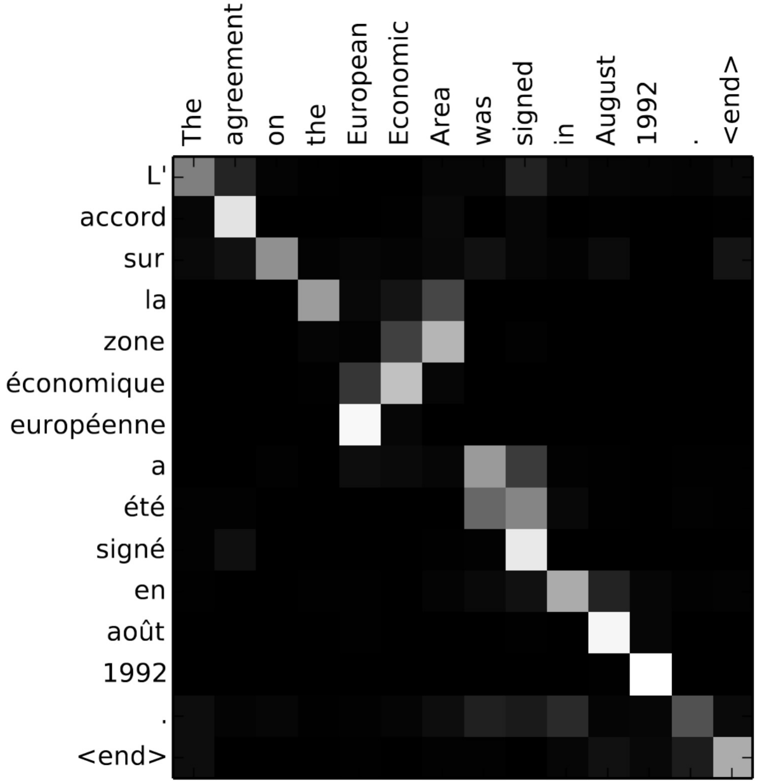


D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *3rd International Conference on Learning Representations (ICLR)*, 2015.

Sequence to Sequence with RNNs + **Attention**

Application: translation

Each pixel shows the weight $\alpha_{t,i}$ of the annotation of the i-th source word for the t-th target word.



D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *3rd International Conference on Learning Representations (ICLR),* 2015.

Sequence to Sequence with RNNs + **Attention**

Application:

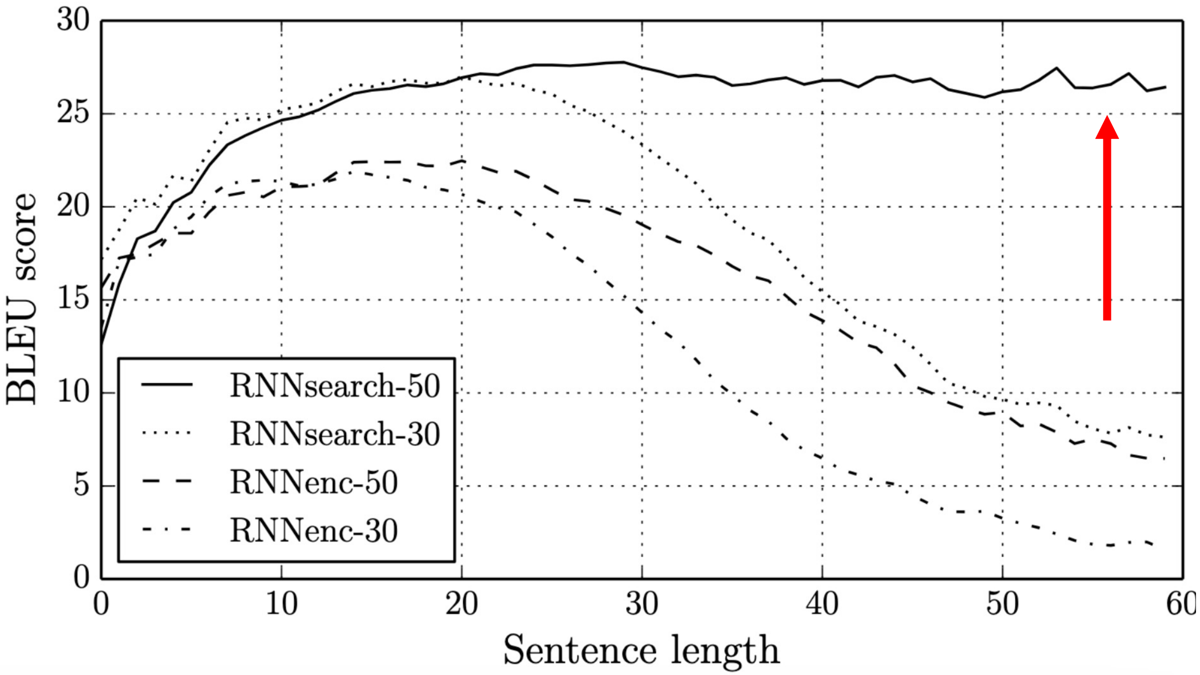
text translation

RNN:

RNNenc

RNN + attention:

RNNsearch



D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” in *3rd International Conference on Learning Representations (ICLR),* 2015.

Motivating Transformers by Understanding the Limitations of Recurrent Models

Challenge 1: Modeling long-range dependencies

Challenge 2: Optimization due to vanishing and exploding gradients

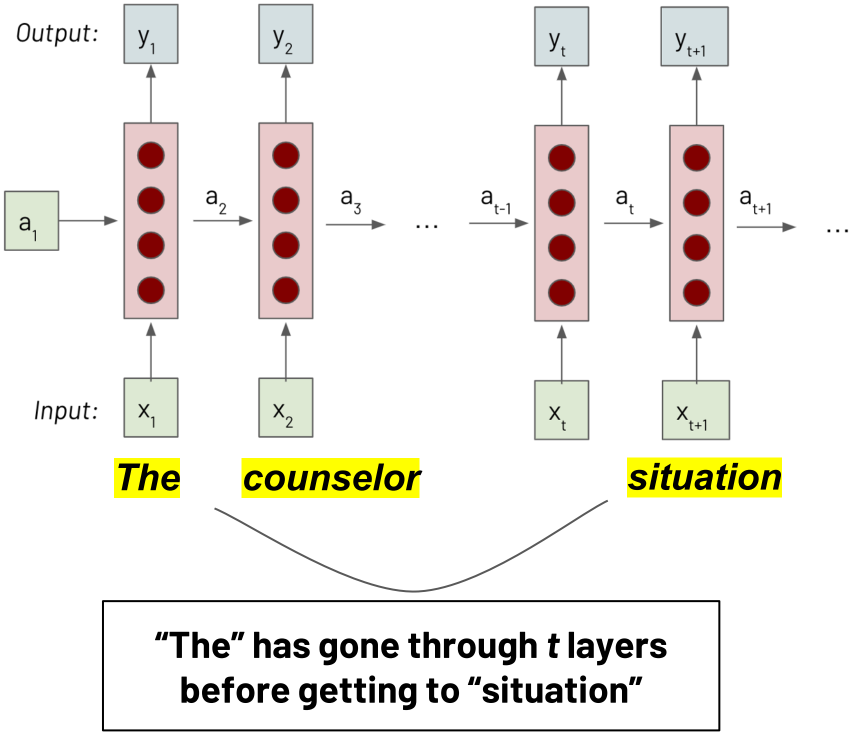
Challenge 3: Slow (sequential, serial) bottleneck

Caveating this discussion: while the challenges we'll discuss originally motivated Transformers, many have continued to make progress on RNNs over the years (S4, Mamba, Linear Attention, GLA, Based, etc.)

Challenge 1: Long Interaction Distances

E.g. "The counselor helped **frame** the situation."

- Performance degrades as the distance between words increases due to memory constraints: Diluted impact of earlier elements on output as sequence progresses

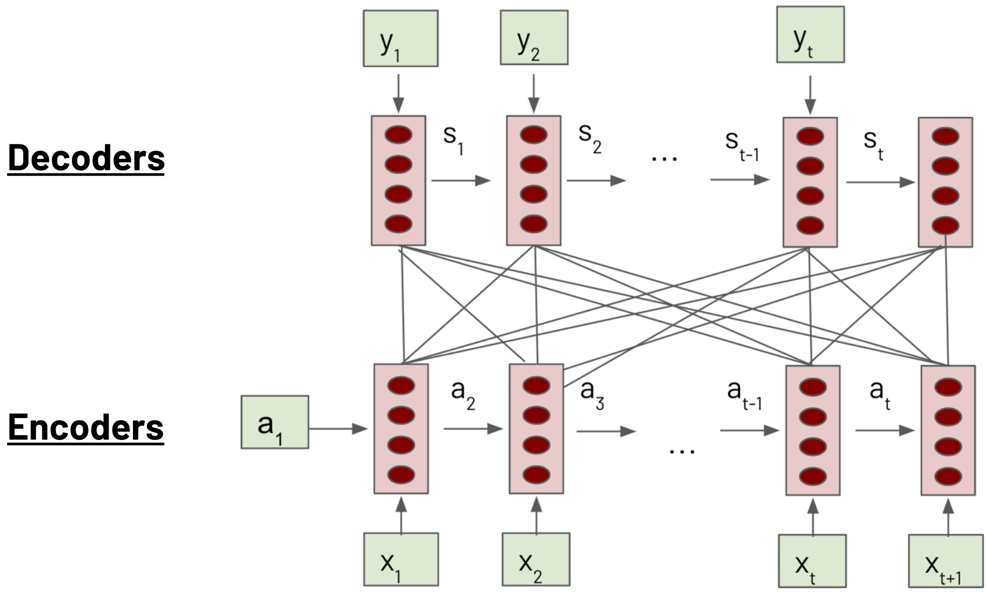


While RNNs + "Attention" has made some progress, the coupling of the "sequential" structure of RNN with Attention still creates difficulties

Key idea of the

RNN + Attention operation:

All tokens interact with all other tokens' representations



Challenge 2: RNNs/ LSTMs are difficult to train!

Backpropagation through many timesteps/"layers"...

Recall: backpropogation is about updating the parameters in a way that reduces the loss. We multiply with respect to each set of parameters at each timestep.

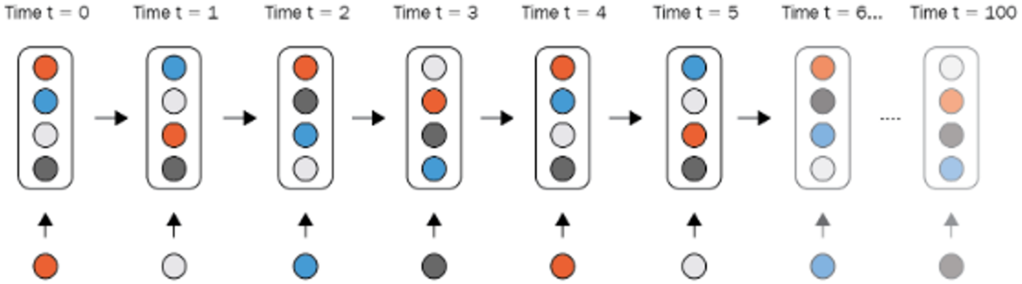
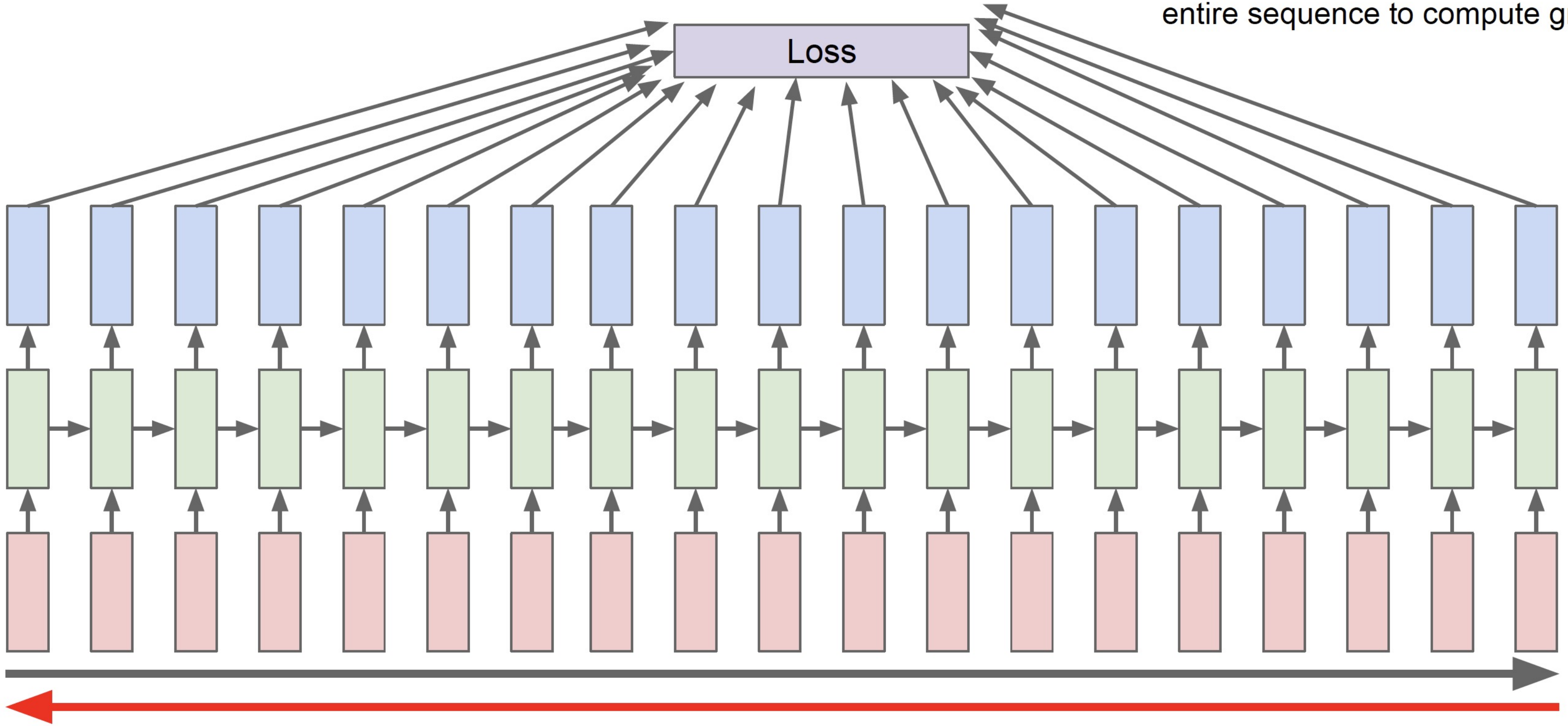


Figure Source: O'Reilly Media

Backpropagation through Time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient



Challenge 2: RNNs/LSTMs are difficult to train!

If the value we are multiplying is **large**, our gradients will grow **exponentially**! The model becomes **unstable!**

If the value we are multiplying is **small**, our gradients will get smaller each timestep, **going to 0**. The network **stops learning/learns too slowly**.

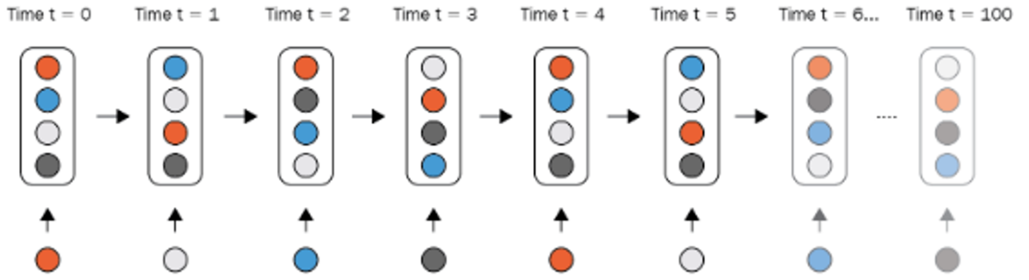
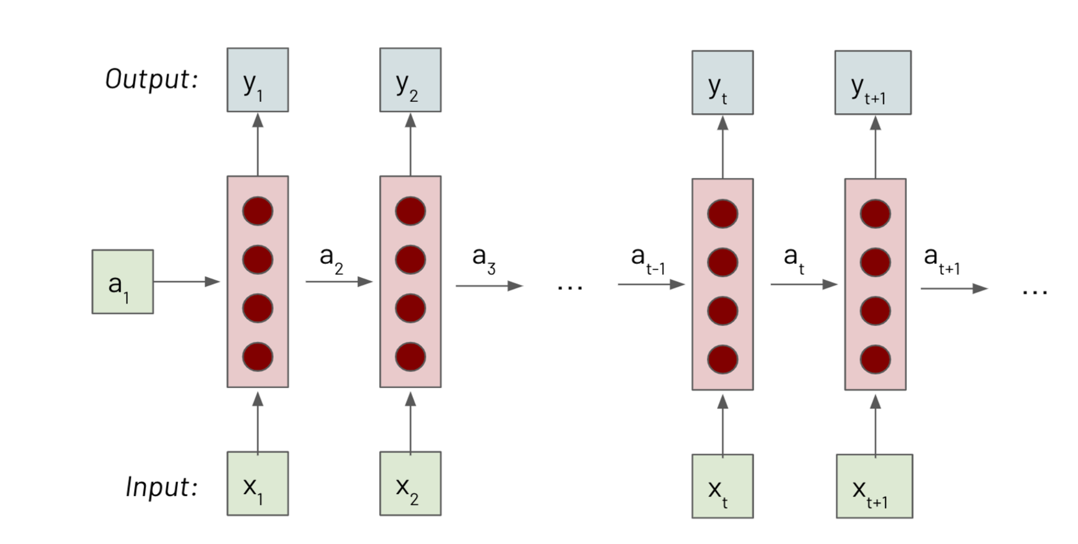


Figure Source: O'Reilly Media

Challenge 3: Parallelizability



Each time step needs to be processed before we can move onto the next step.

Decoupling “Attention” from RNNs

- **Recall:** attention determines the **importance of elements** to be passed forward in the model.
  - These weights lets the model pay **attention** to the **most significant parts**

- **Objective**: a more general attention mechanism not confined to RNNs
  - We need a modified procedure to:

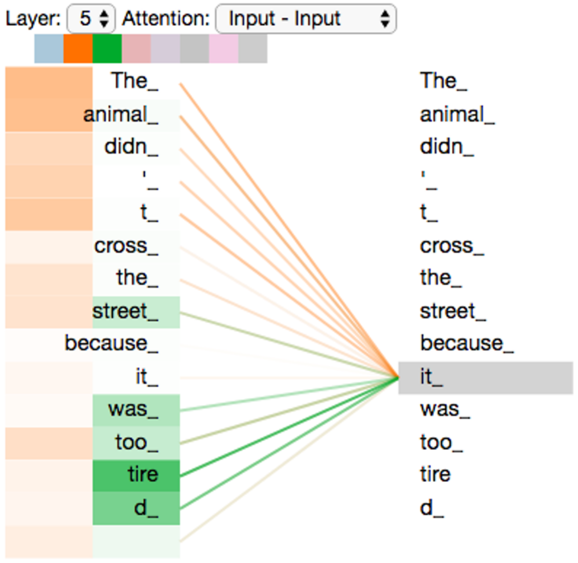
1. Determine weights based on context that indicate the elements to attend to
2. Apply these weights to enhance attended features

Self-Attention and Transformers

- Allows to “focus attention” on particular aspects of the input while generating the output.
- Done by using a set of parameters, called "weights," that determine how much attention should be paid to each input token at each time step.
- These weights are computed using a combination of the input and the current hidden state of the model.

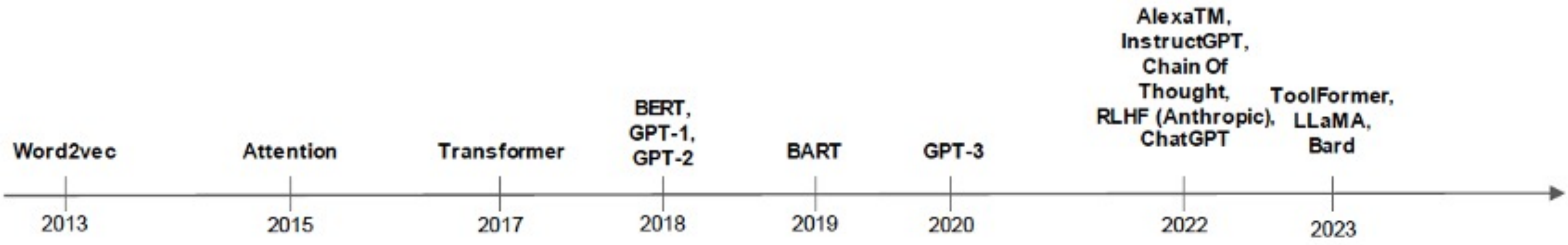
$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

A. Vaswani et al. Attention Is All You Need. NeurIPS 2017.



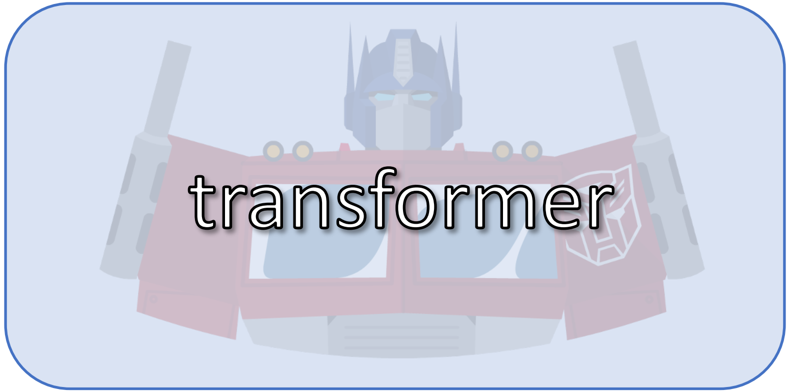
In encoding the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired". The model's representation of the word "it" thus bakes in some of the representation of both "animal" and "tired".
https://jalammar.github.io/illustrated-transformer/

Transformers – the current ”standard” for building LLMs and Foundation Models



Slides for video from:
Prof. Jia-Bin Huang
University of Maryland, College Park

https://www.youtube.com/watch?v=rcWMRA9E5RI

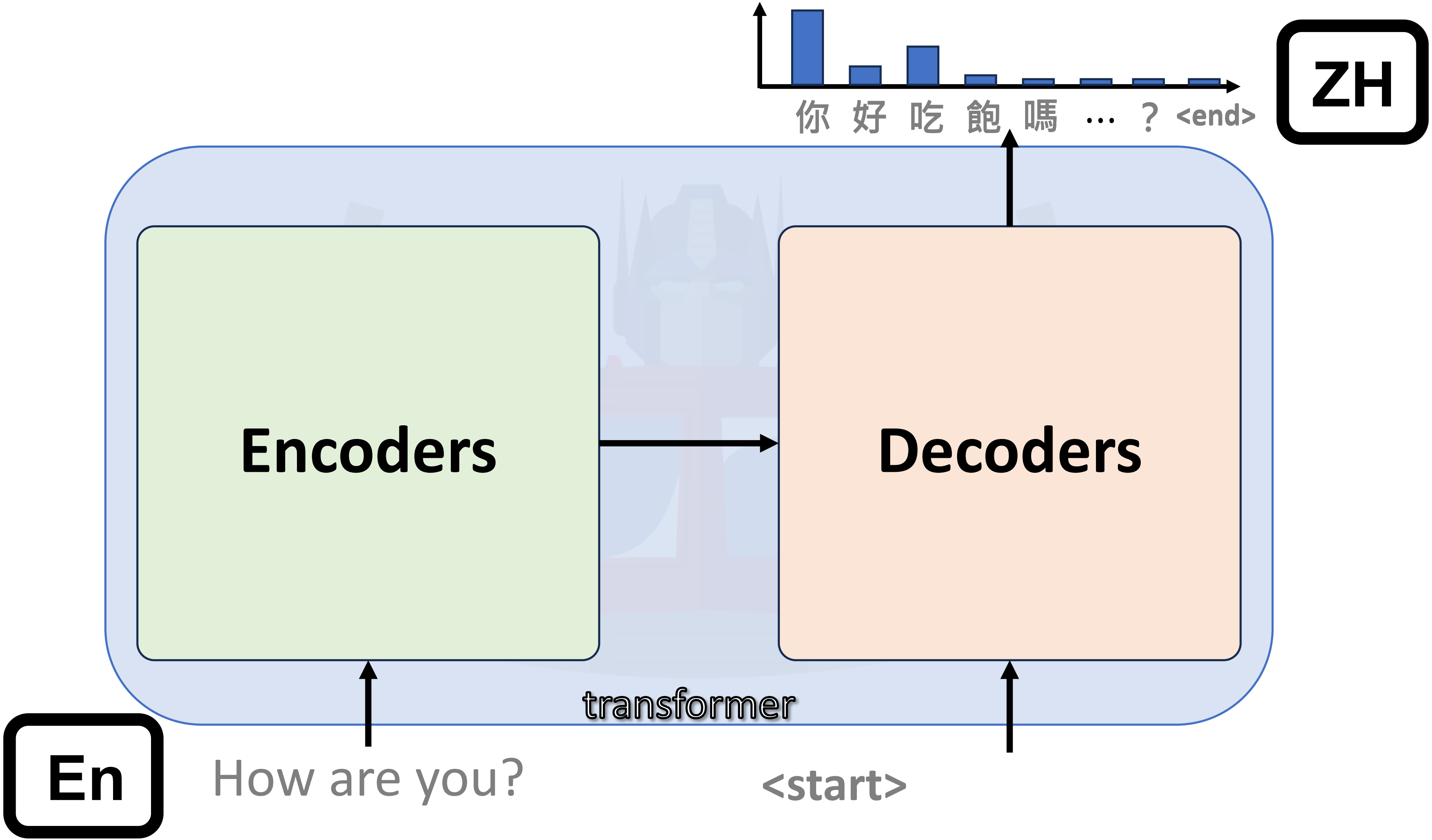


🤔 **What?**

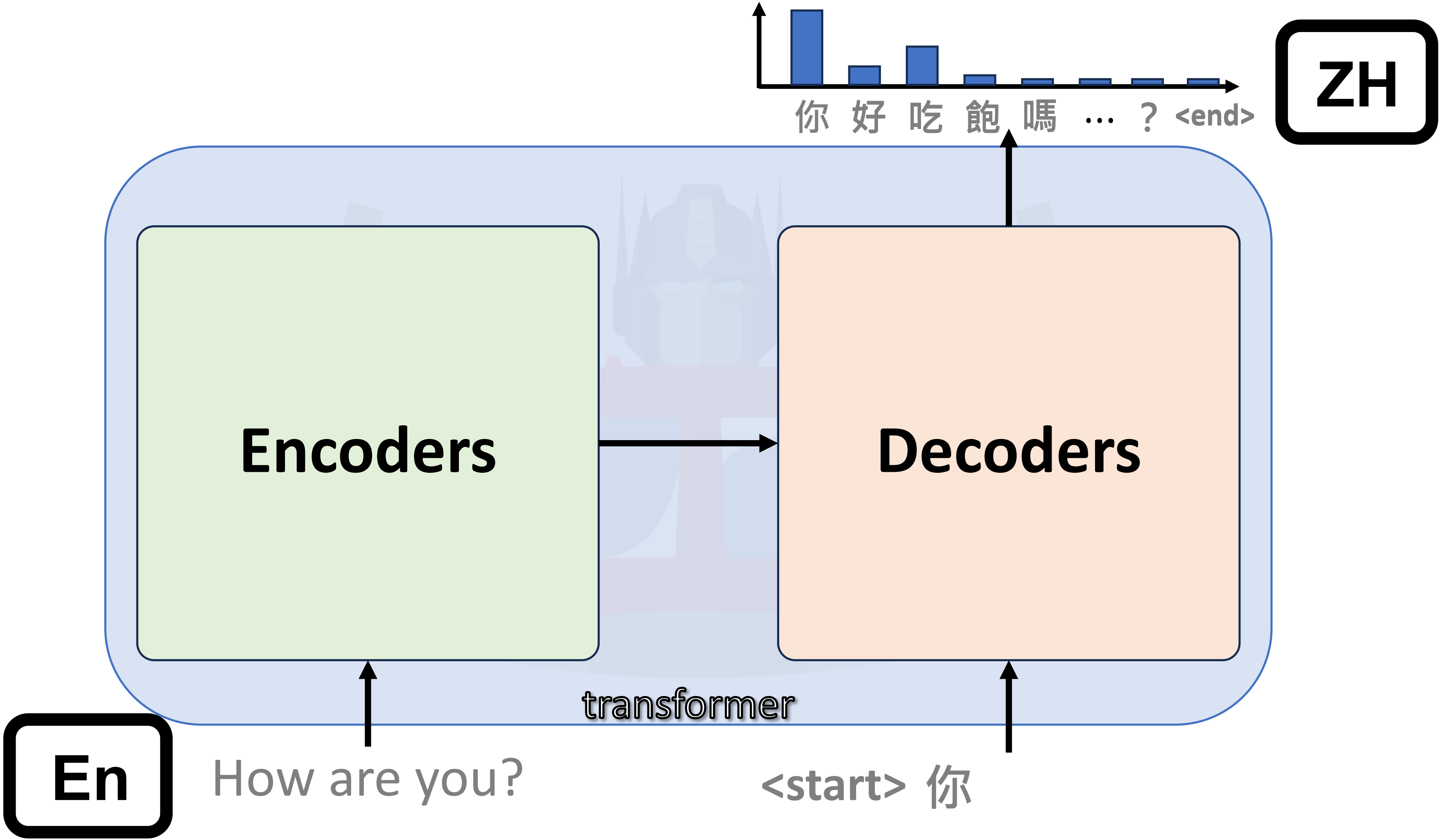
🤔 **How?**

🤔 **Why?**

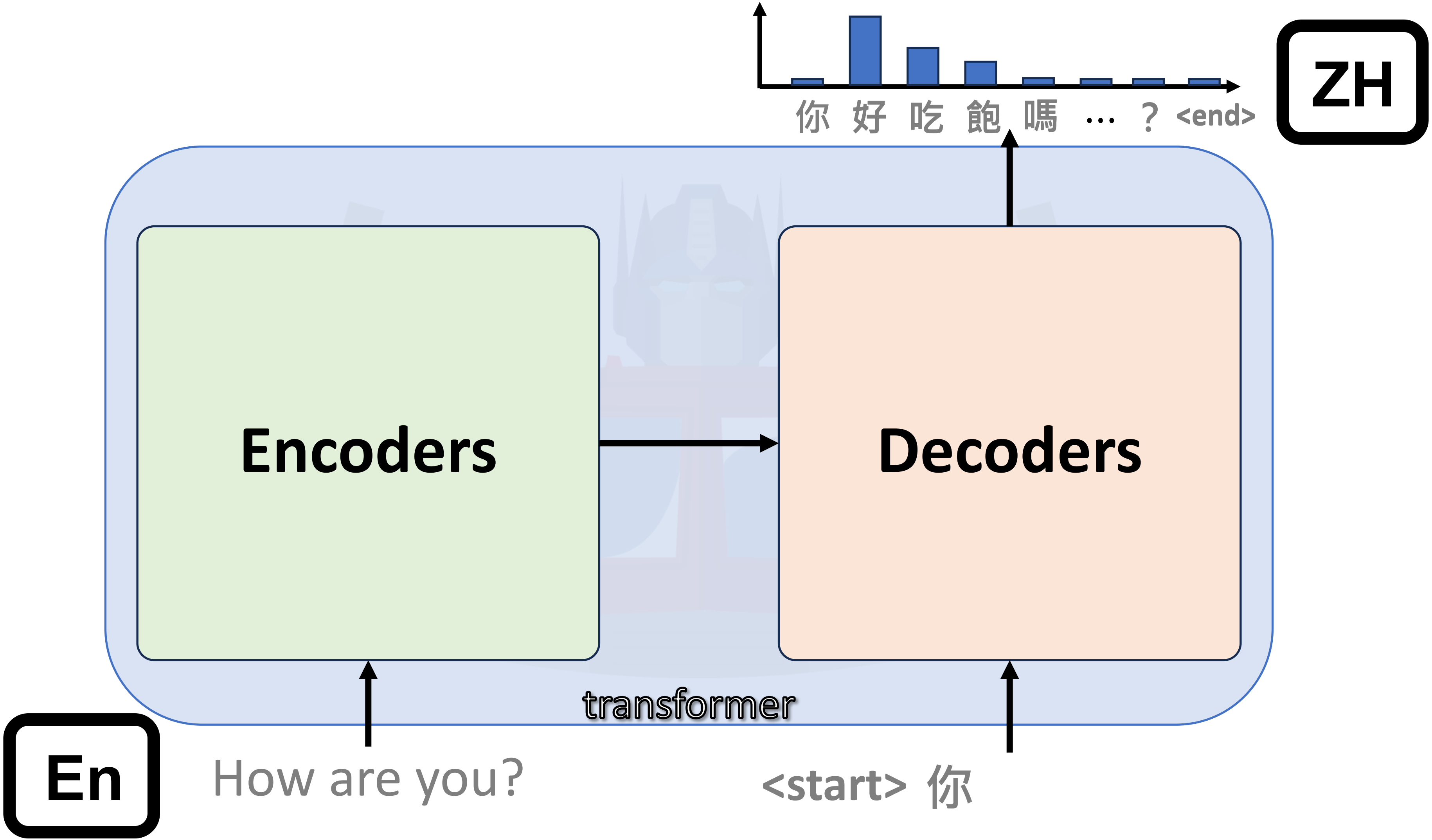
Sequence-to-Sequence model

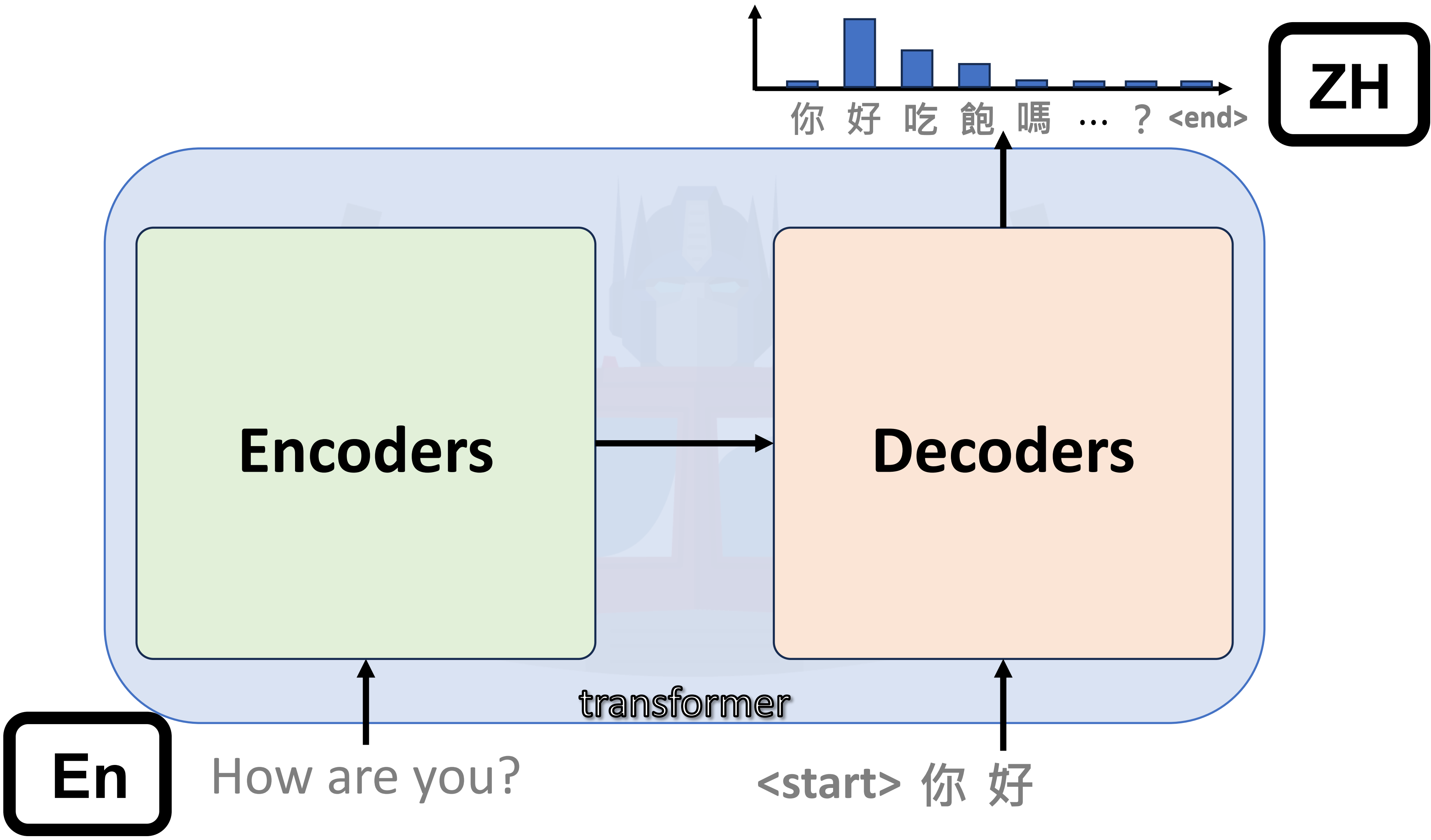


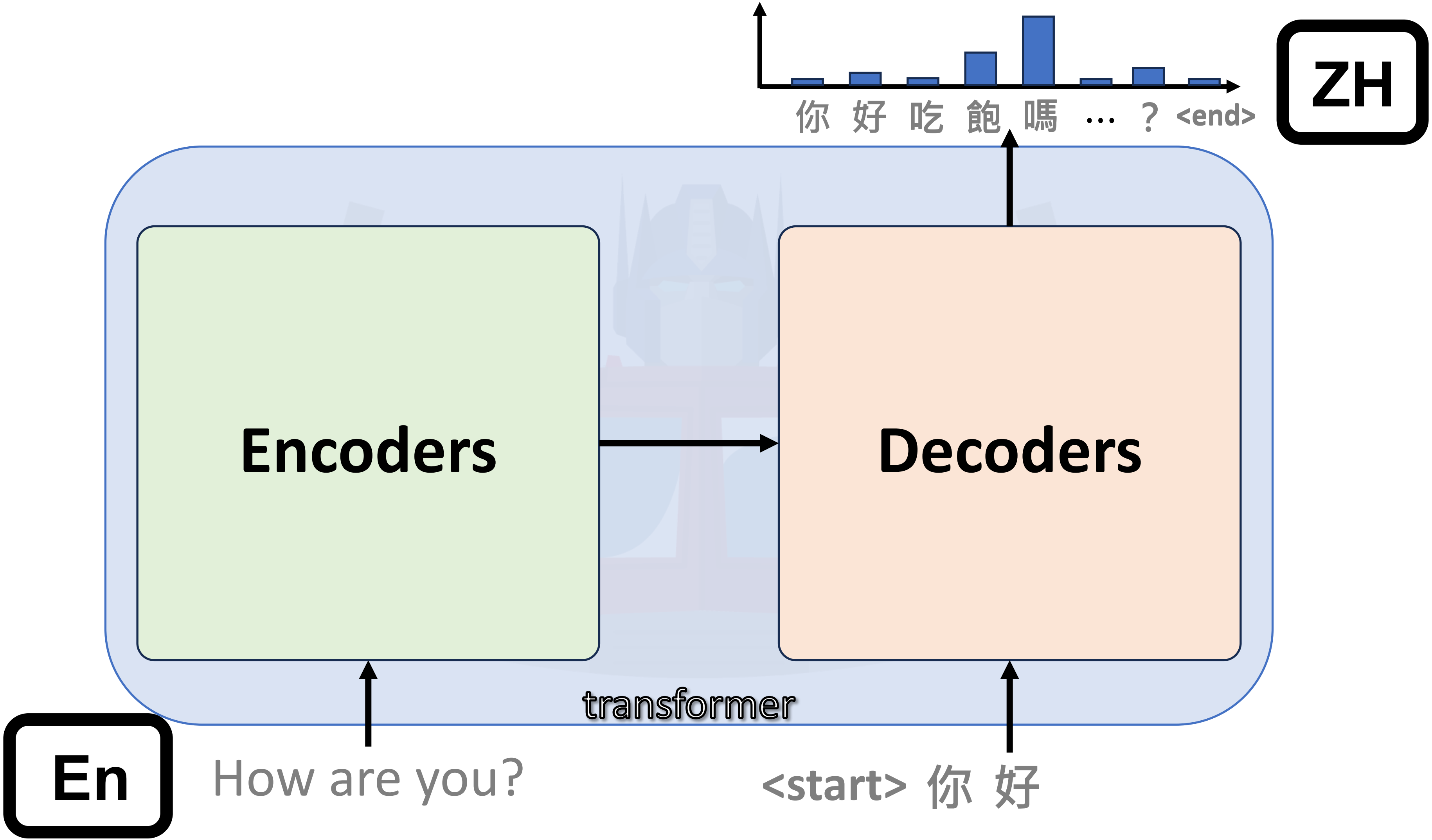
你 好 吃 飽 嗎 … ? <end>
ZH
Encoders
Decoders
transformer
En
How are you?
<start>



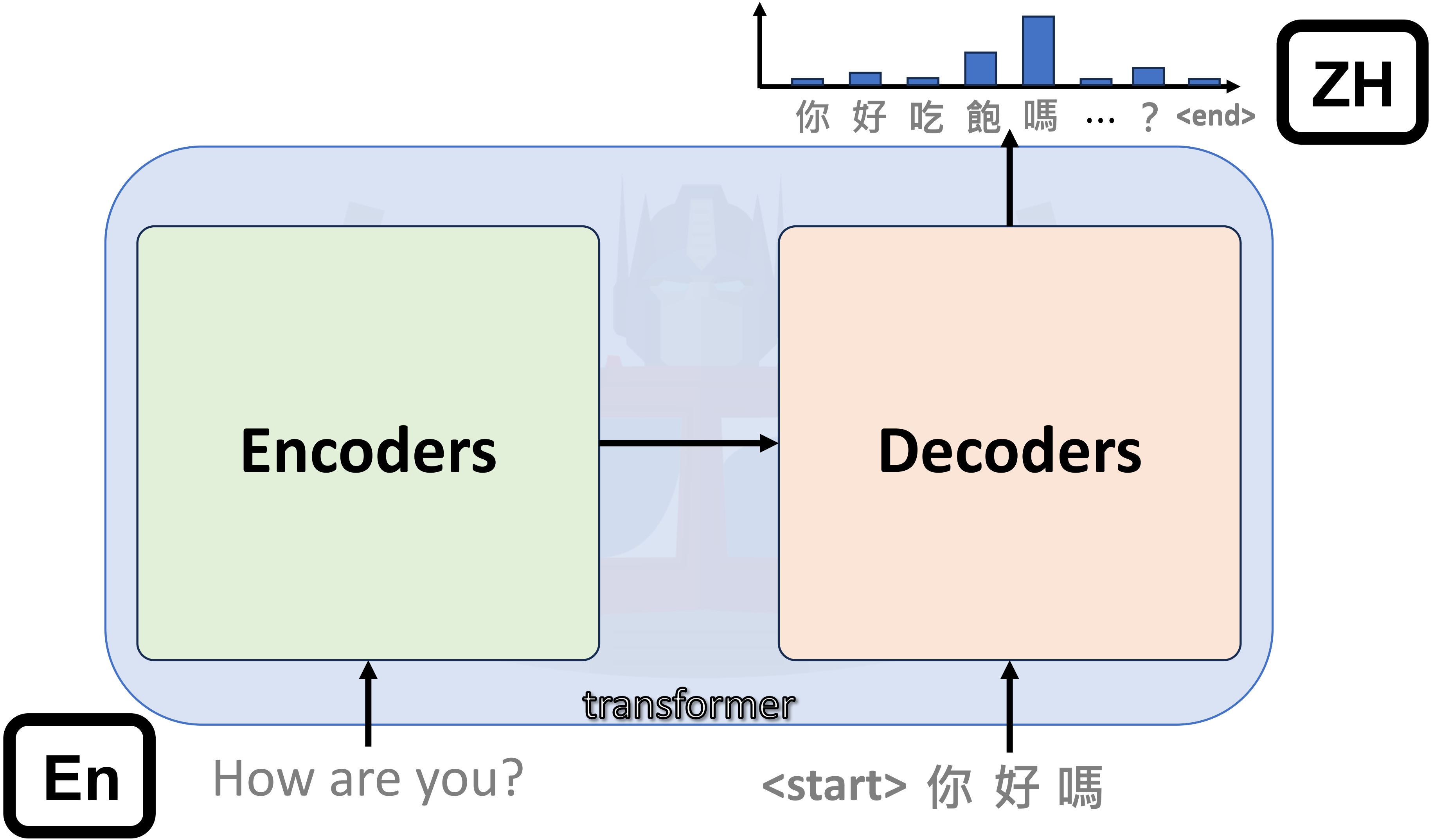
你 好 吃 飽 嗎 … ? <end>
ZH
Encoders
Decoders
transformer
En
How are you?
<start> 你

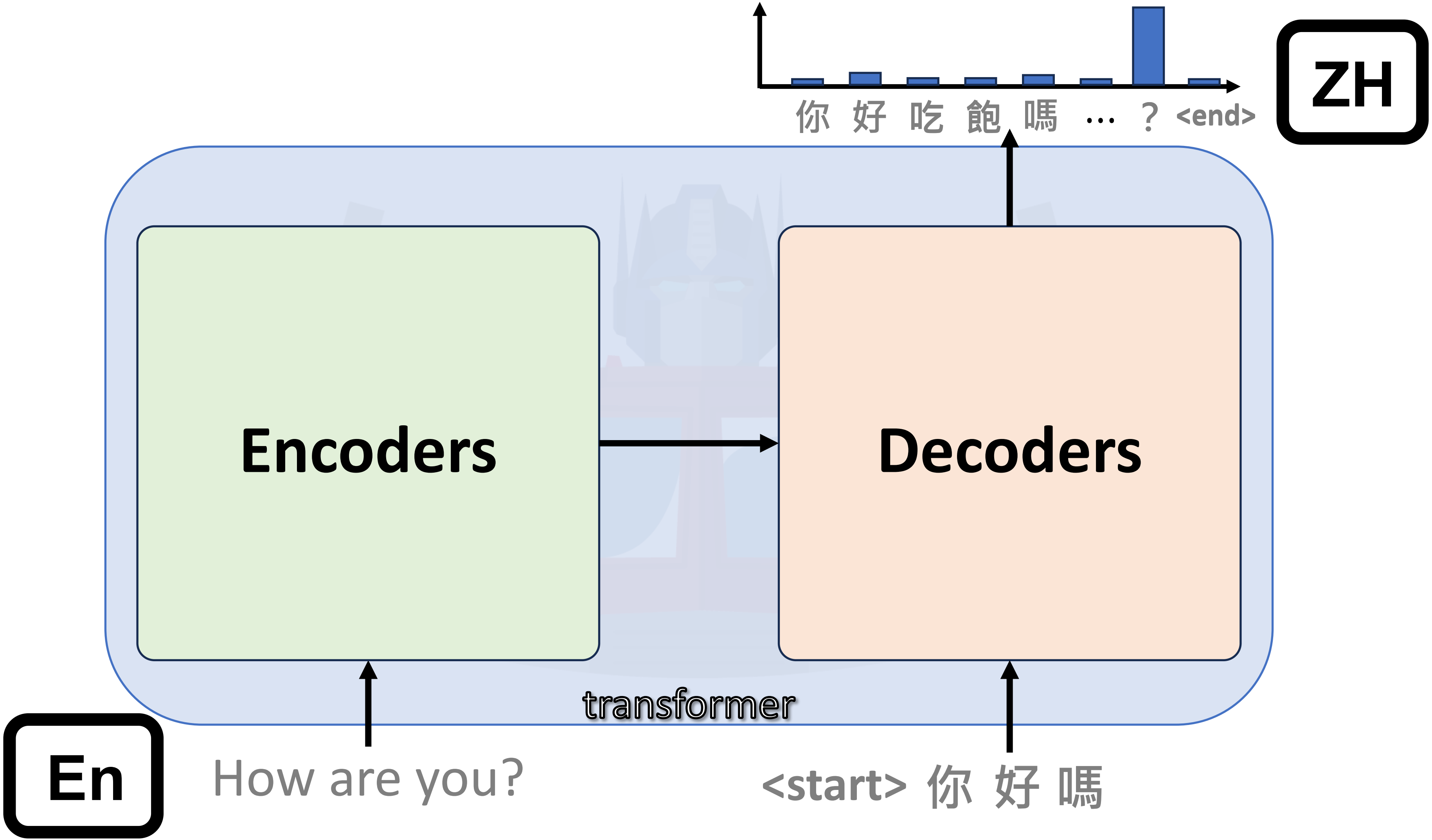

你 好 吃 飽 嗎 … ? <end>
ZH
Encoders
Decoders
transformer
En
How are you?
<start> 你


你 好 吃 飽 嗎 … ? <end>
ZH
Encoders
Decoders
transformer
En
How are you?
<start> 你 好

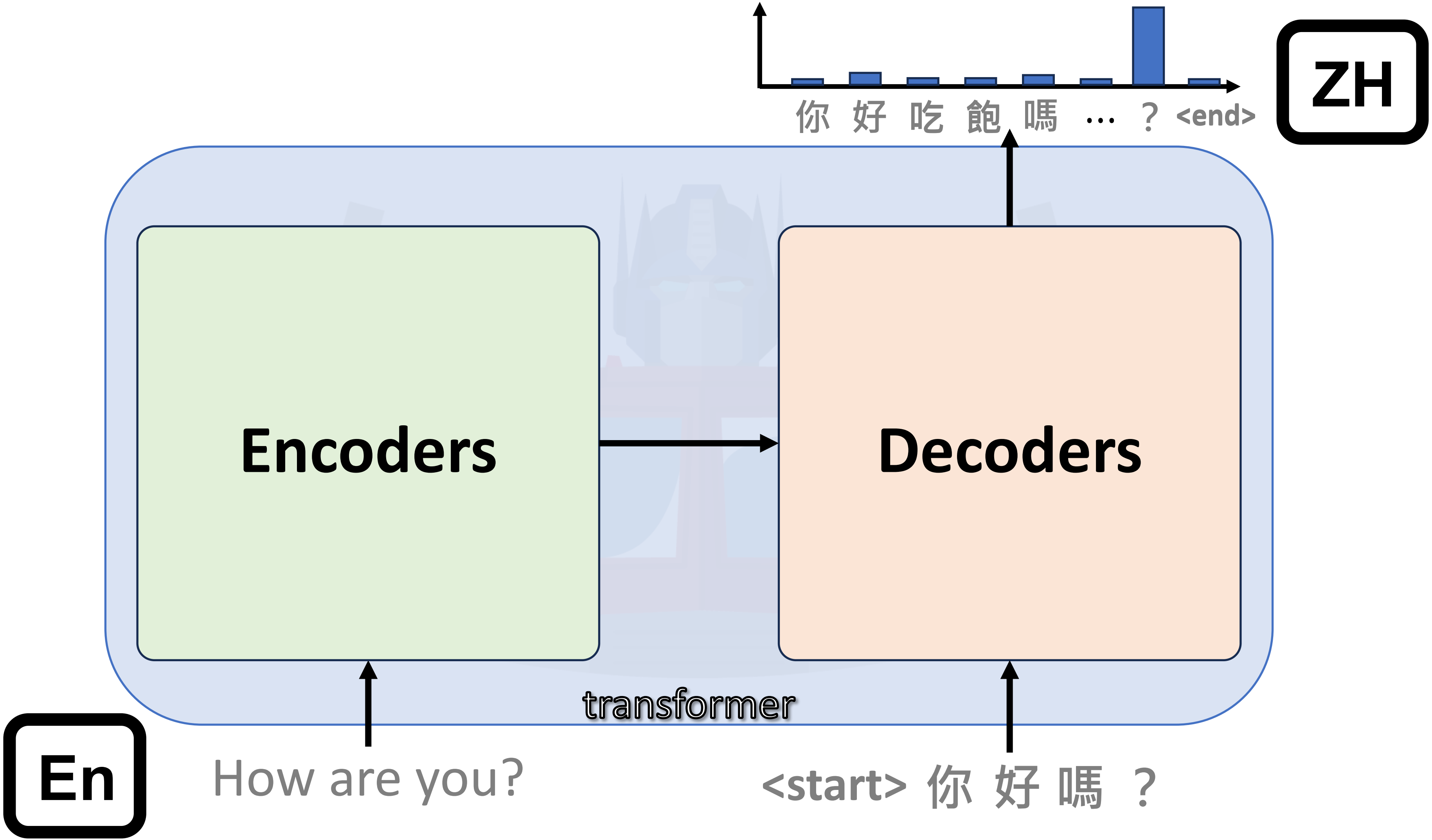


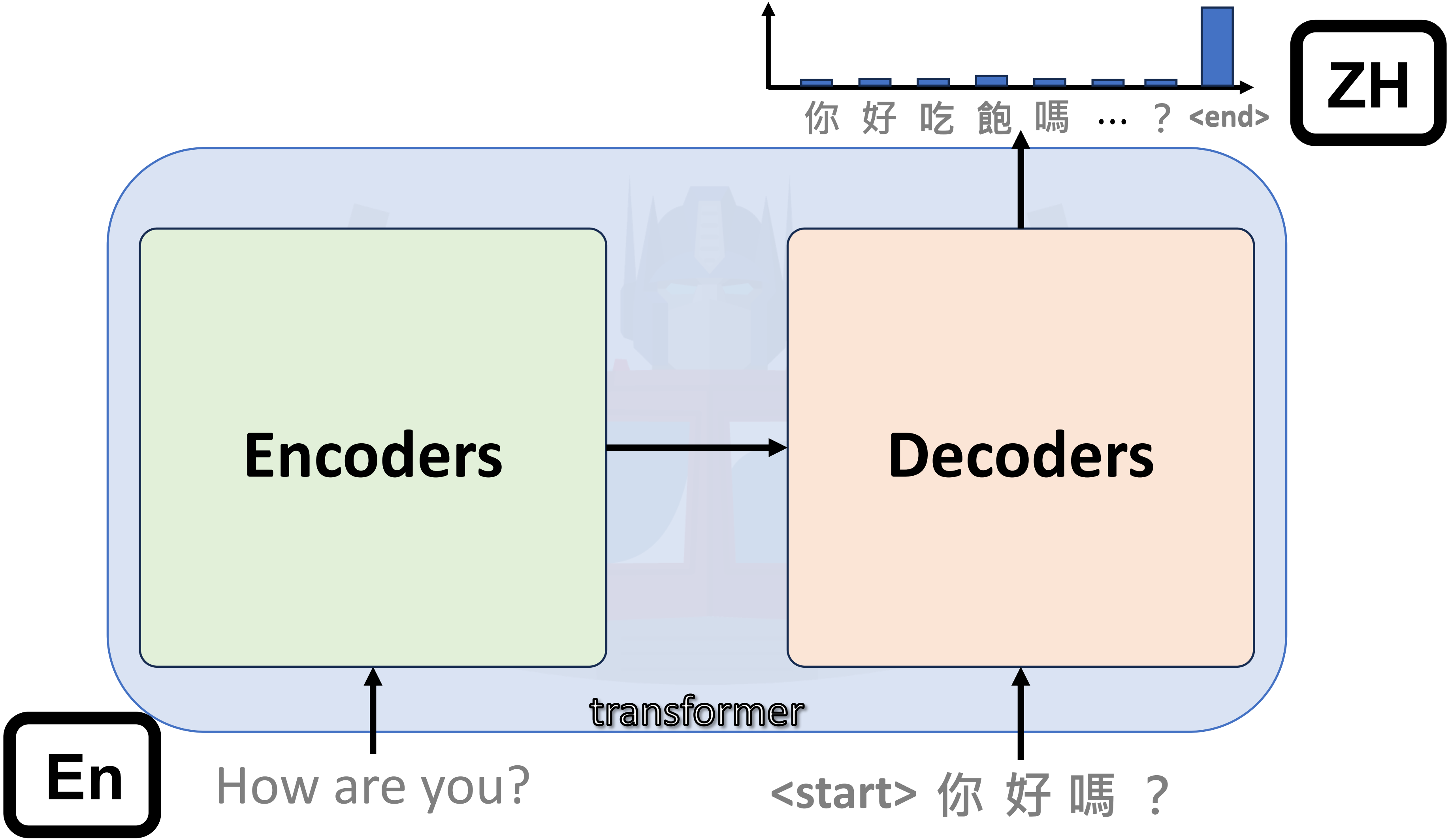
你 好 吃 飽 嗎 … ? <end>
ZH
Encoders
Decoders
transformer
En
How are you?
<start> 你 好


ZH
你 好 吃 飽 嗎 … ? <end>
Encoders
Decoders
transformer
En
How are you?
<start> 你 好 嗎

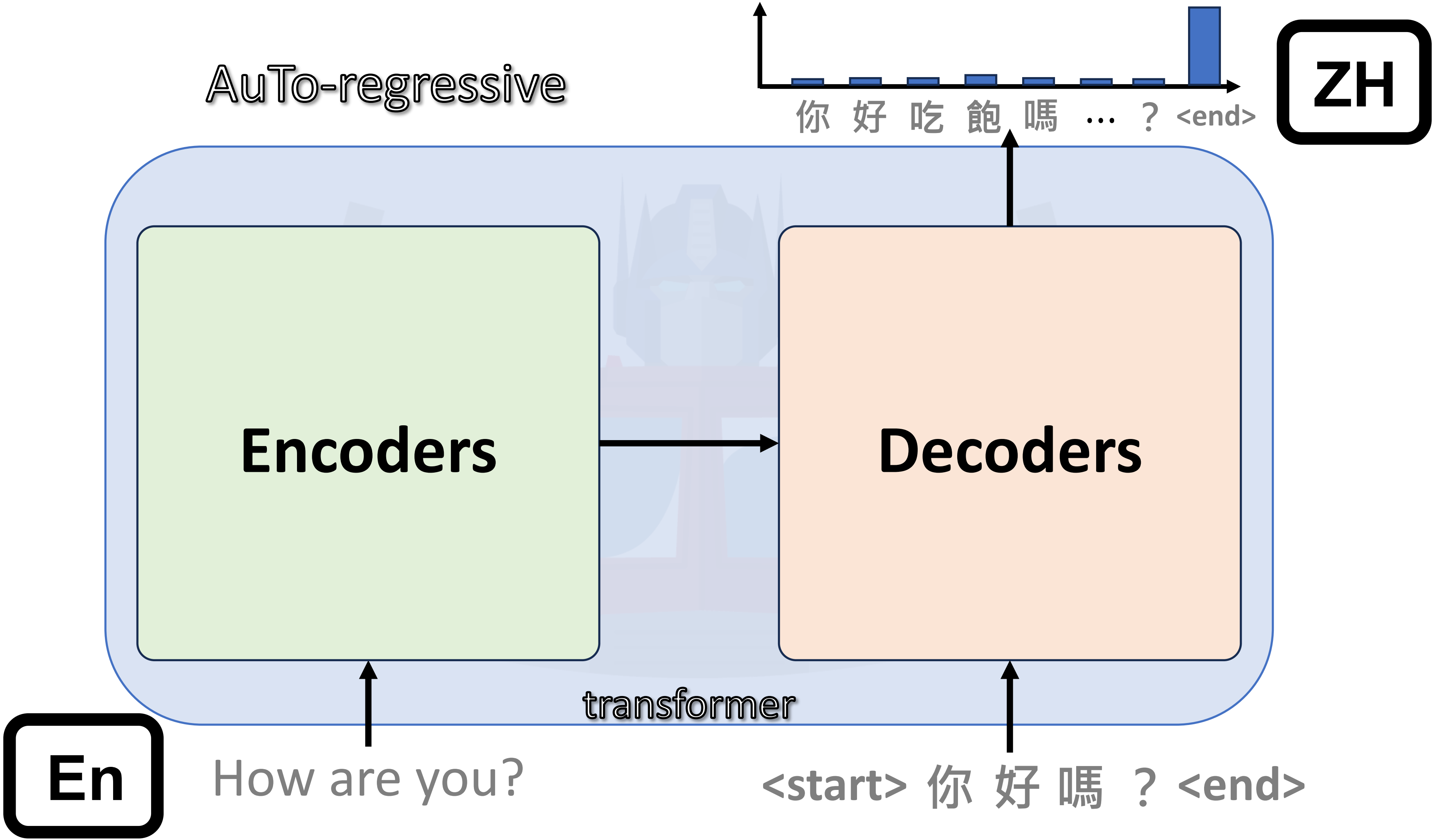


你 好 吃 飽 嗎 … ? <end>
ZH
Encoders
Decoders
transformer
En
How are you?
<start> 你 好 嗎

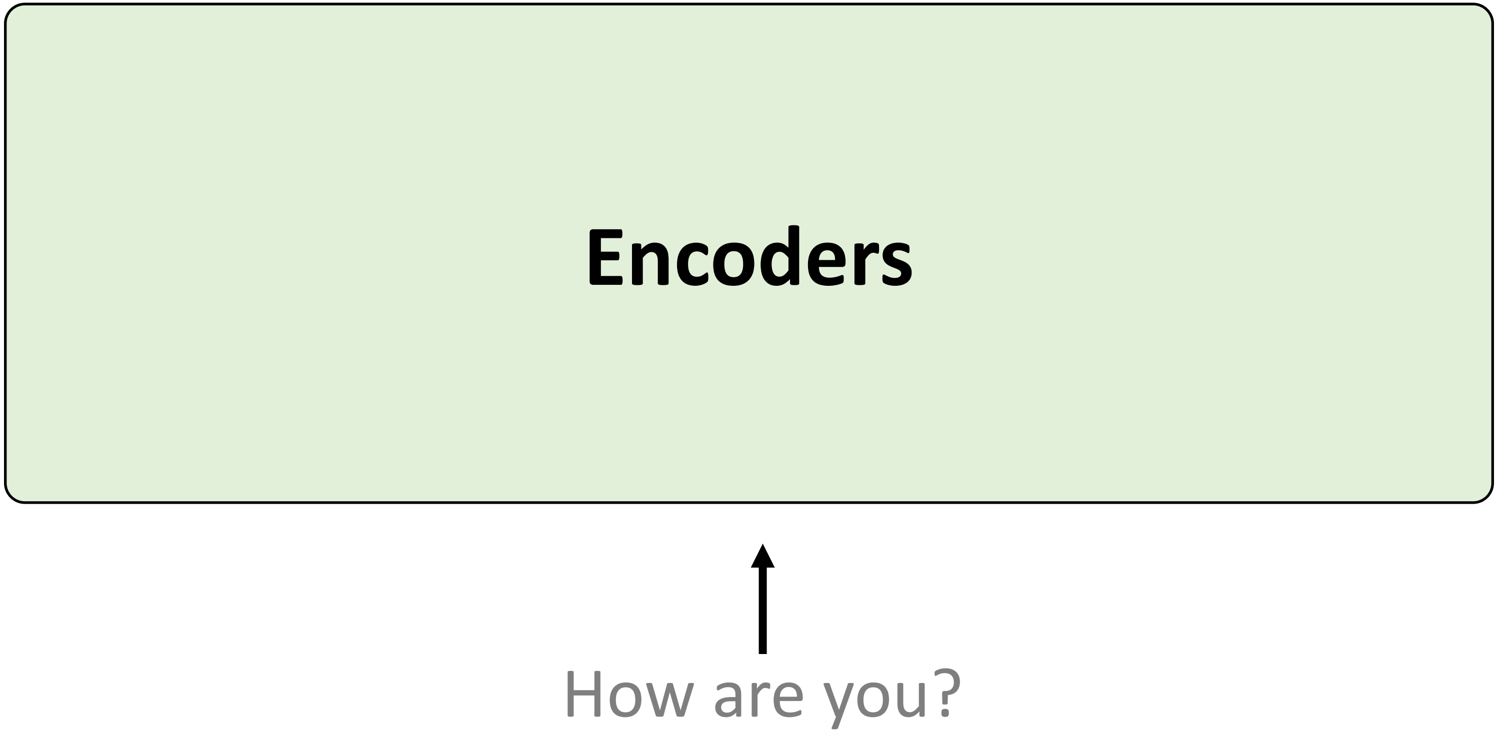

你 好 吃 飽 嗎 … ? <end>
ZH
Encoders
Decoders
transformer
En
How are you?
<start> 你 好 嗎 ？



你 好 吃 飽 嗎 … ? <end>
ZH
Encoders
Decoders
transformer
En
How are you?
<start> 你 好 嗎 ？



AuTo-regressive
你 好 吃 飽 嗎 … ? <end>
ZH
Encoders
Decoders
transformer
En
How are you?
<start> 你 好 嗎 ? <end>


Encoders
How are you?

Tokenization

Many	words	map	to	one	token	,	but	some	don	'	t:	indivisible	.
8607	4339	2472	311	832	4037	11	719	1063	1541	956	25	3687	23936	13

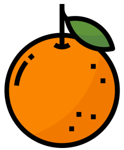
One-hot encoding

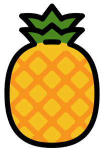
$$
\# \text{ tokens} \quad
\begin{array}{ccccc}
\text{cat} & \text{dog} & \text{bear} & \text{cow} & \text{indiv} \\
\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} &
\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} &
\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} &
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} &
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}
\end{array}
$$

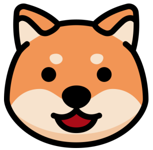
Value 1 at 3687th entry

TOKEN EMBEDDING

One-hot encoding



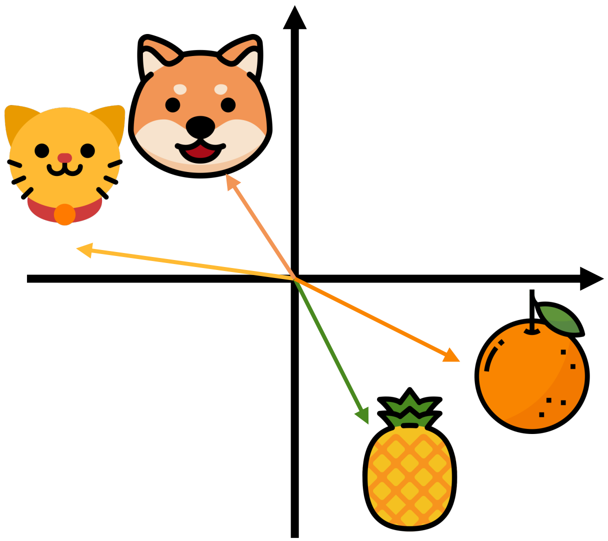




$$\begin{array}{ccccc} \text{cat} & \text{dog} & \text{bear} & \text{cow} & \text{indiv} \\ \begin{bmatrix}1\\0\\0\\0\\0\\\vdots\\0\end{bmatrix} & \begin{bmatrix}0\\1\\0\\0\\0\\\vdots\\0\end{bmatrix} & \begin{bmatrix}0\\0\\1\\0\\0\\\vdots\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\1\\0\\\vdots\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\0\\0\\\vdots\\0\end{bmatrix} \end{array}$$

Value 1 at 3687th entry

TOKEN EMBEDDING

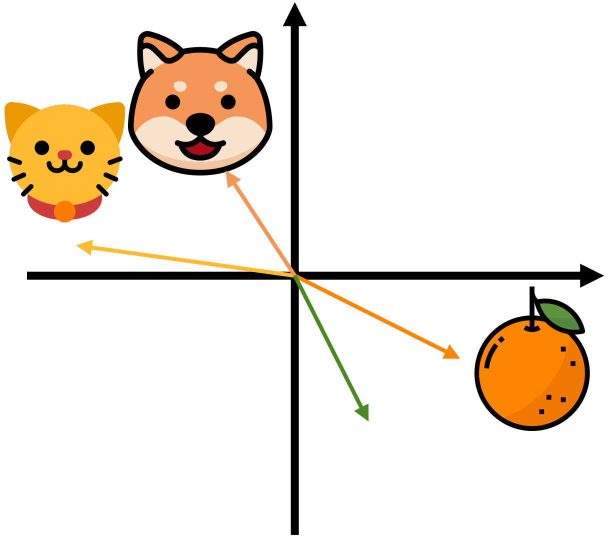


Embedding Space

$$
\begin{array}{ccccc}
\text{cat} & \text{dog} & \text{bear} & \text{cow} & \text{indiv} \\
\begin{bmatrix}1\\0\\0\\0\\0\\\vdots\\0\end{bmatrix} &
\begin{bmatrix}0\\1\\0\\0\\0\\\vdots\\0\end{bmatrix} &
\begin{bmatrix}0\\0\\1\\0\\0\\\vdots\\0\end{bmatrix} &
\begin{bmatrix}0\\0\\0\\1\\0\\\vdots\\0\end{bmatrix} &
\begin{bmatrix}0\\0\\0\\0\\0\\\vdots\\0\end{bmatrix}
\end{array}
$$

Value 1 at 3687th entry

TOKEN EMBEDDING



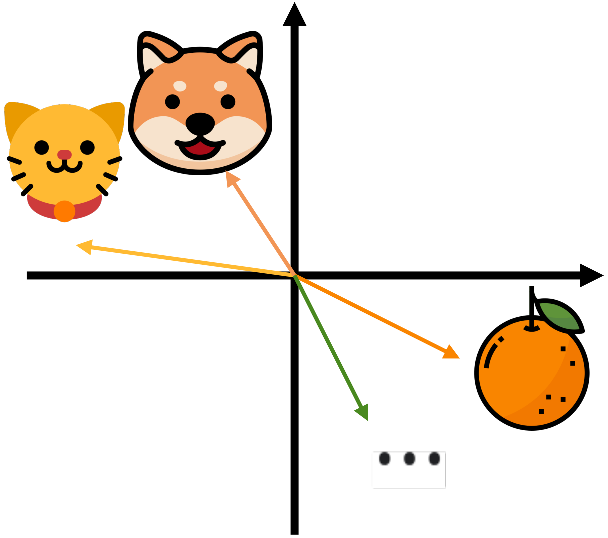
Embedding Space

$$
d \begin{bmatrix} 0.5 \\ 2.7 \\ 1.2 \\ \vdots \\ 0.2 \end{bmatrix} = d \overset{\#\ \text{tokens}}{\begin{bmatrix} \quad \boldsymbol{W_E} \quad \end{bmatrix}} \begin{array}{c} \text{dog} \\ \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \end{array}
$$

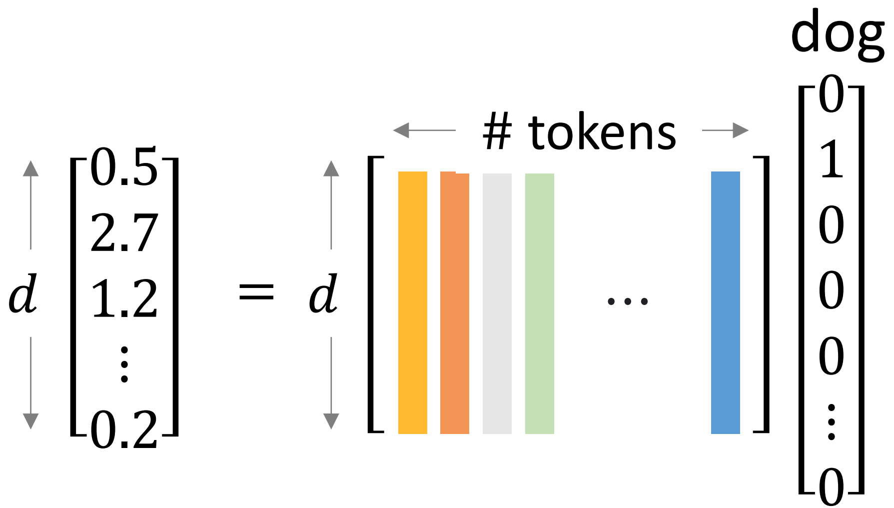
Embedded token

Embedding Matrix

TOKEN EMBEDDING



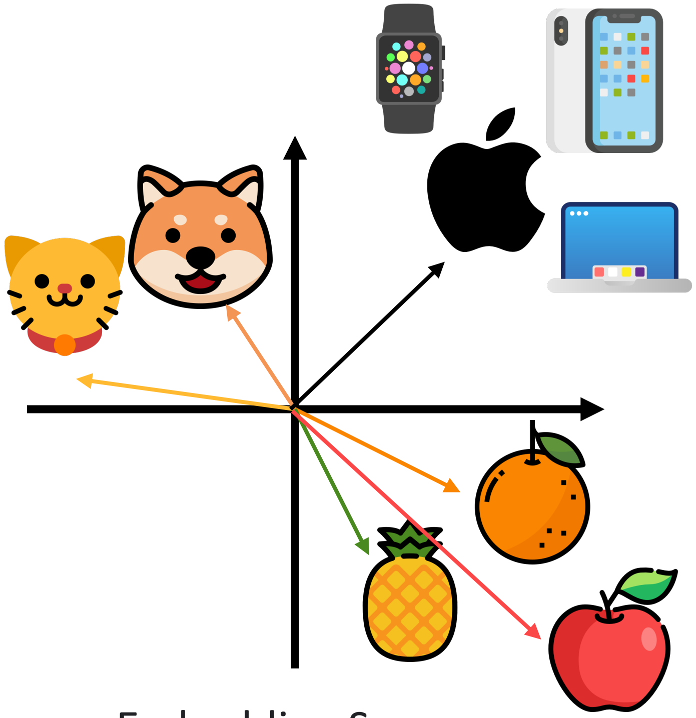
Embedding Space



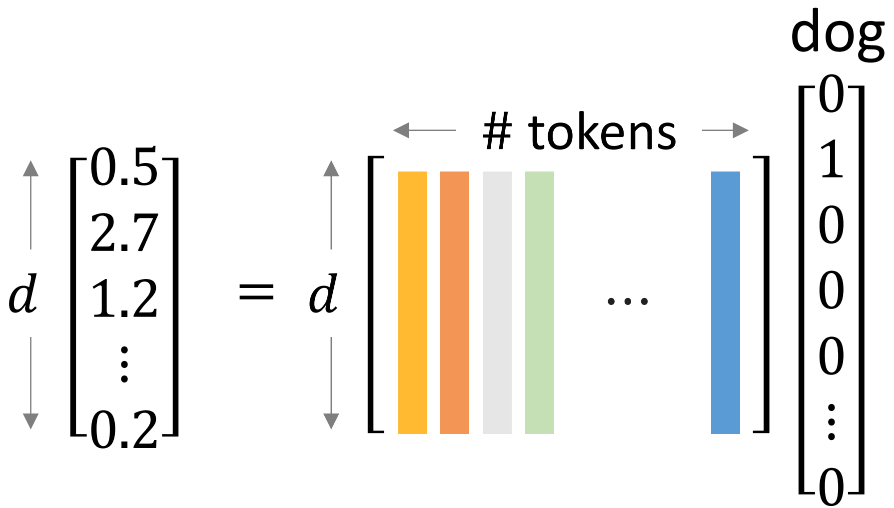
Embedded token

Embedding Matrix

TOKEN EMBEDDING



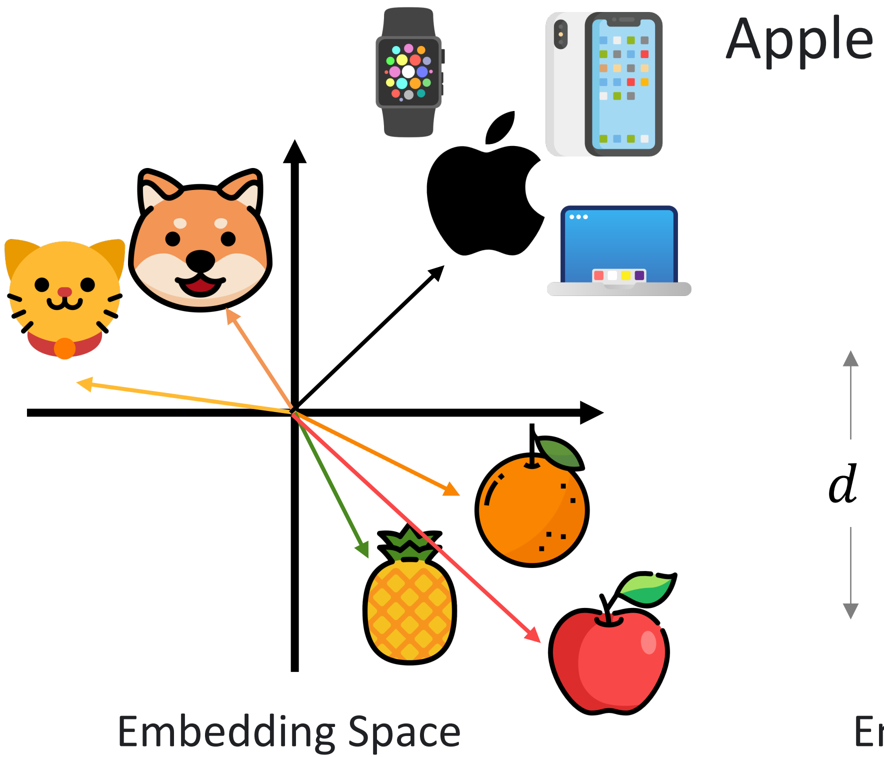
Embedding Space

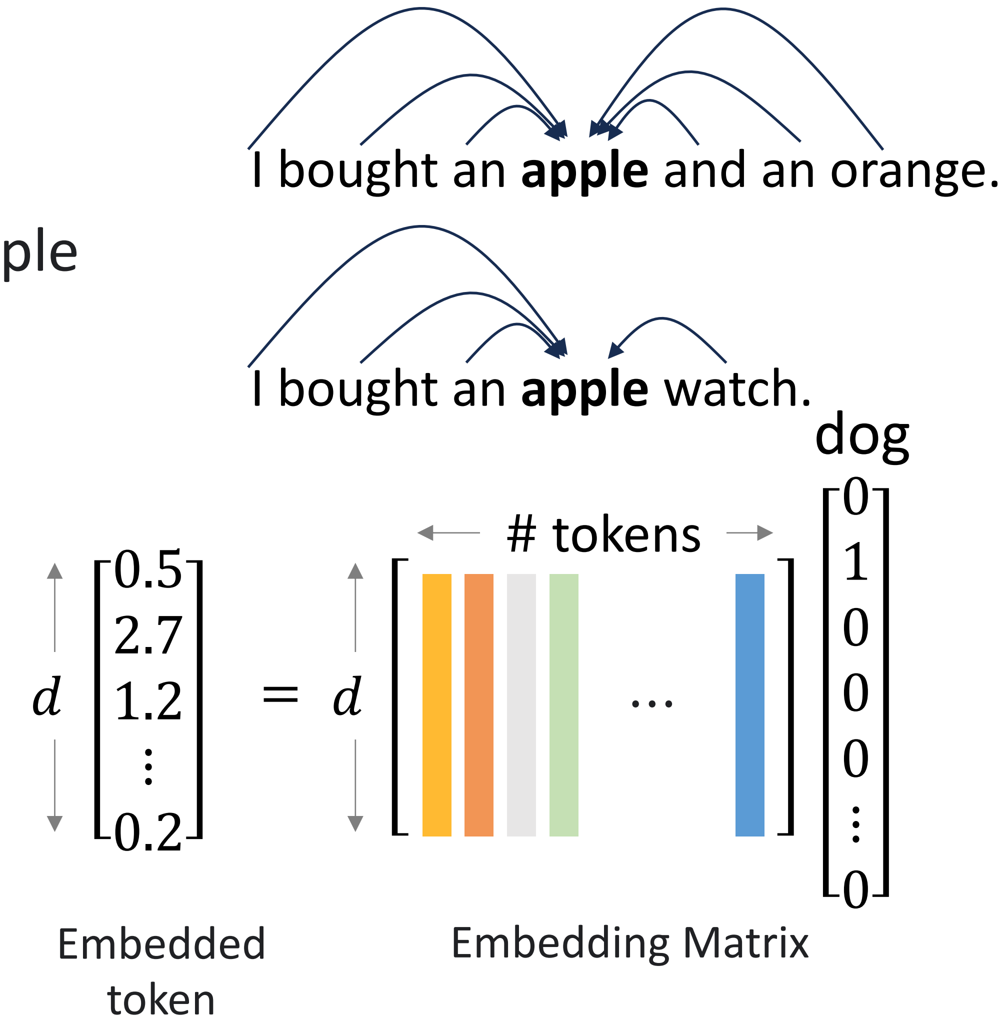


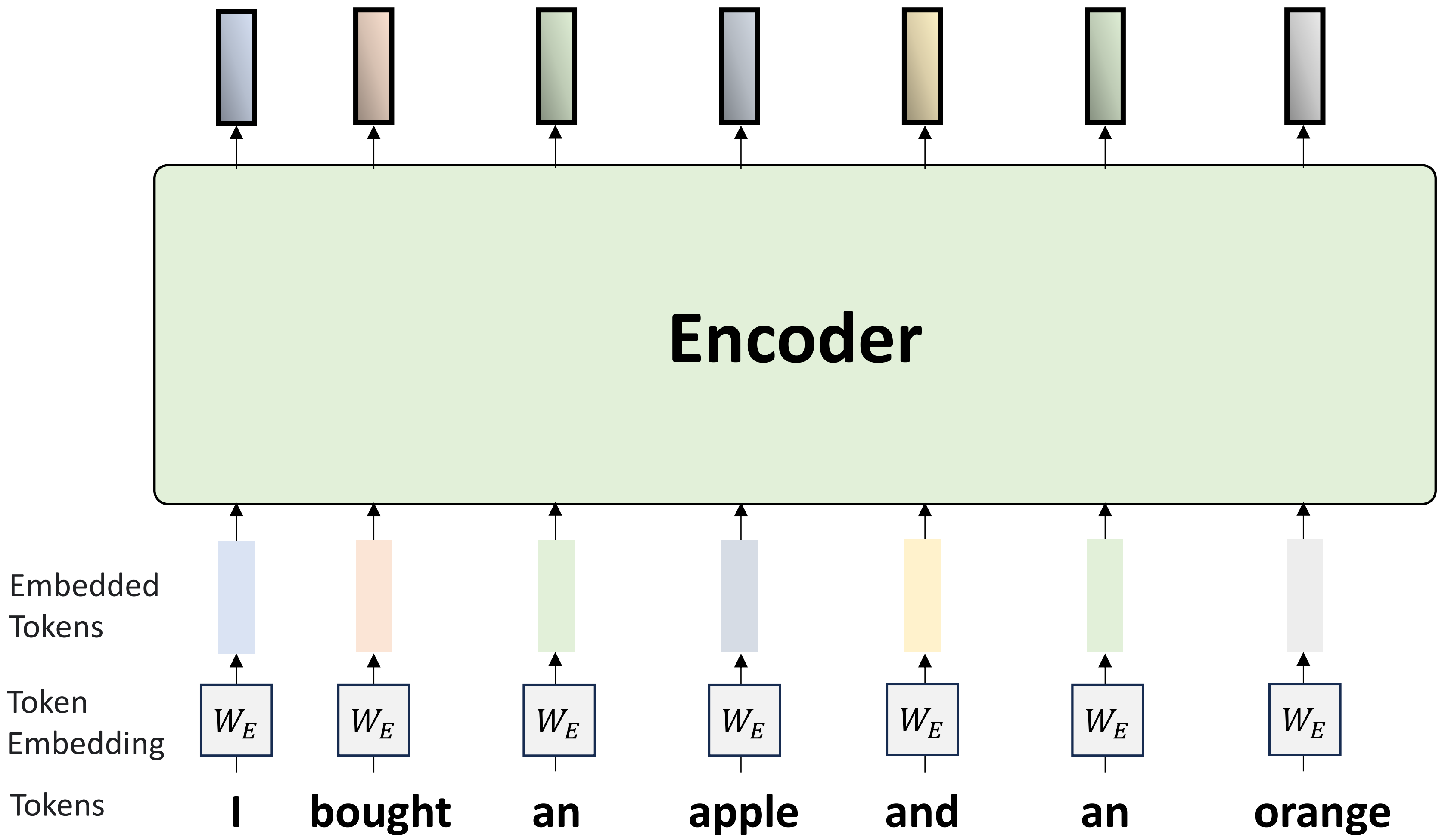
Embedded token

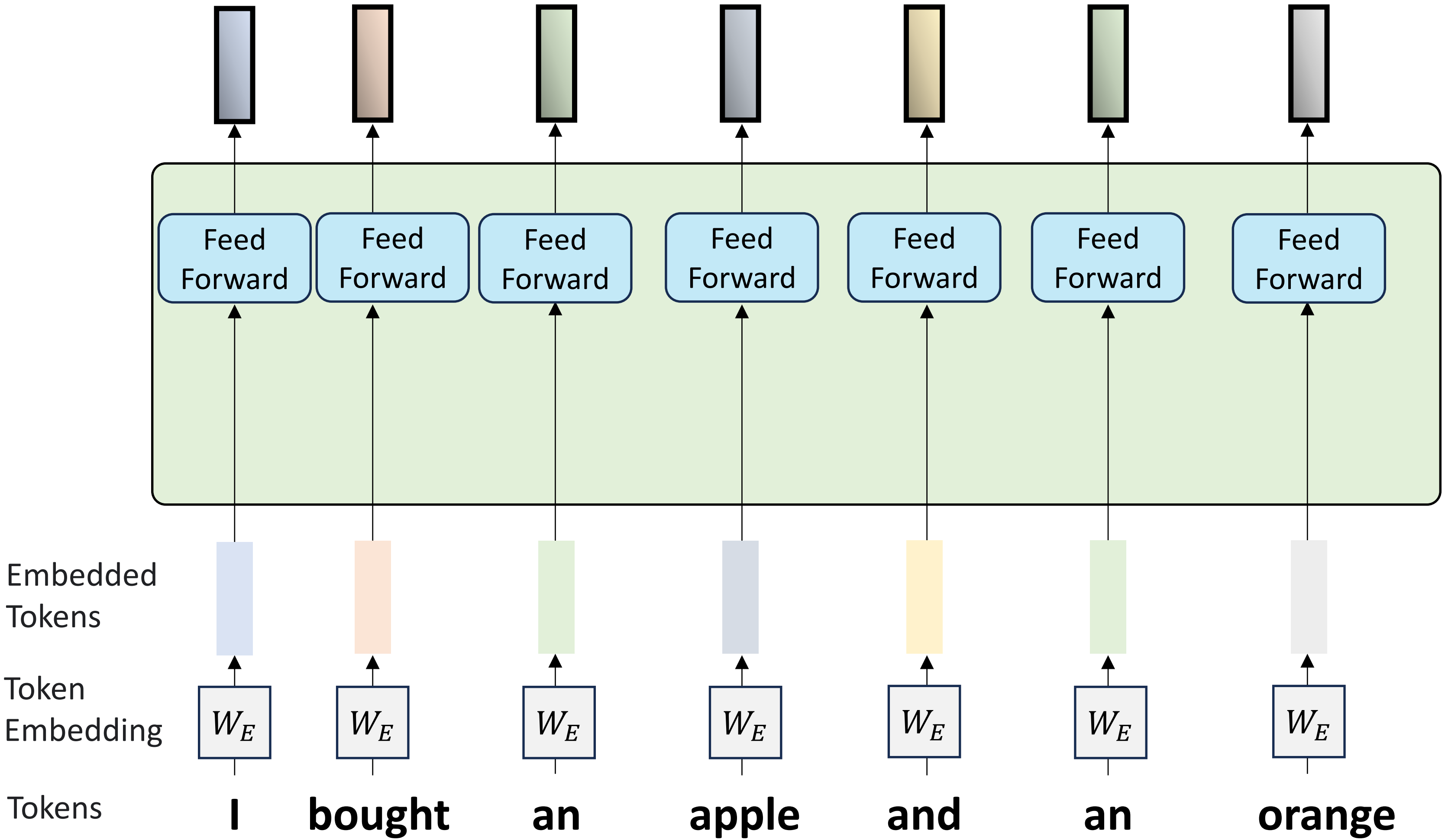
Embedding Matrix

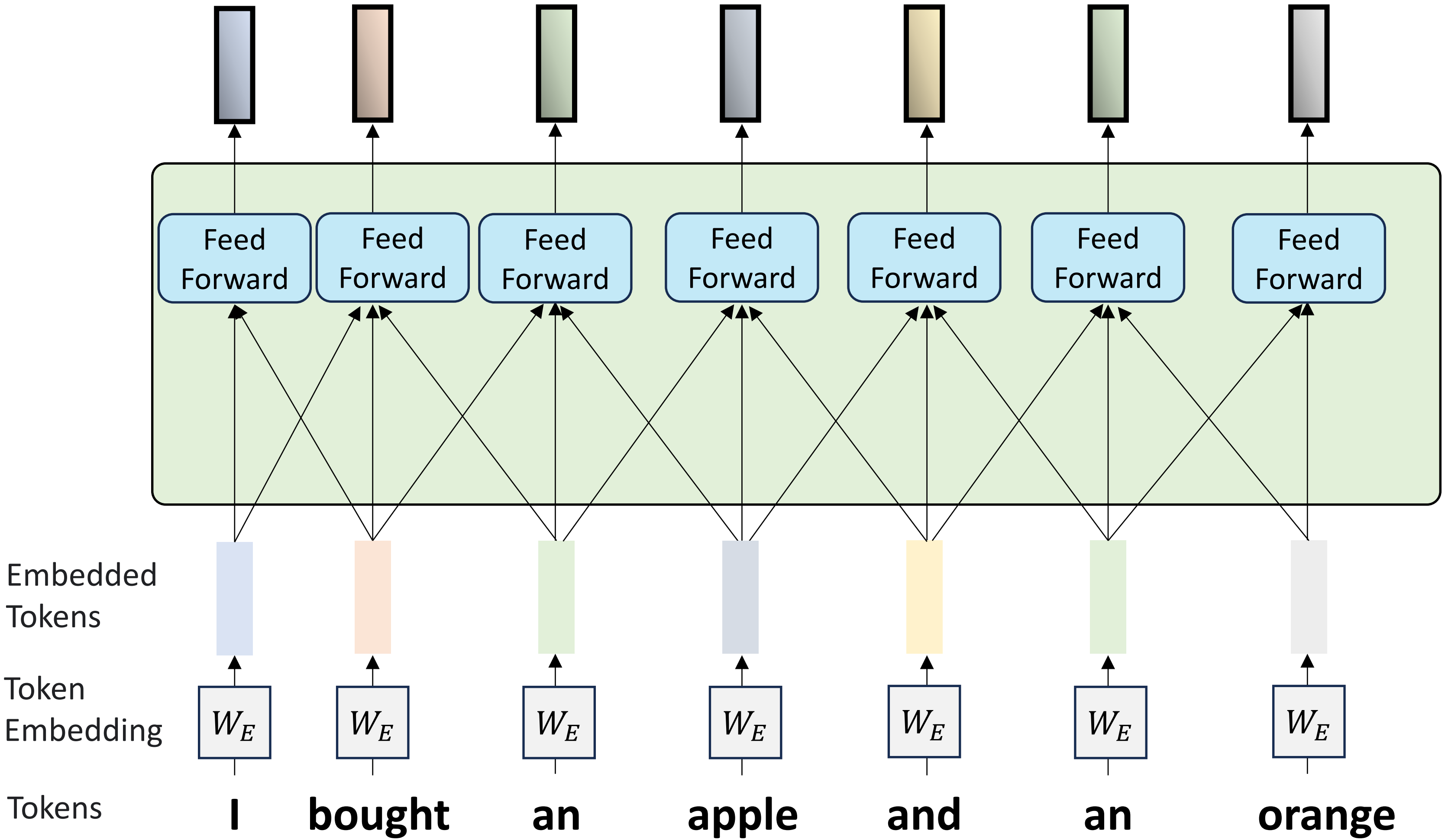
TOKEN EMBEDDING

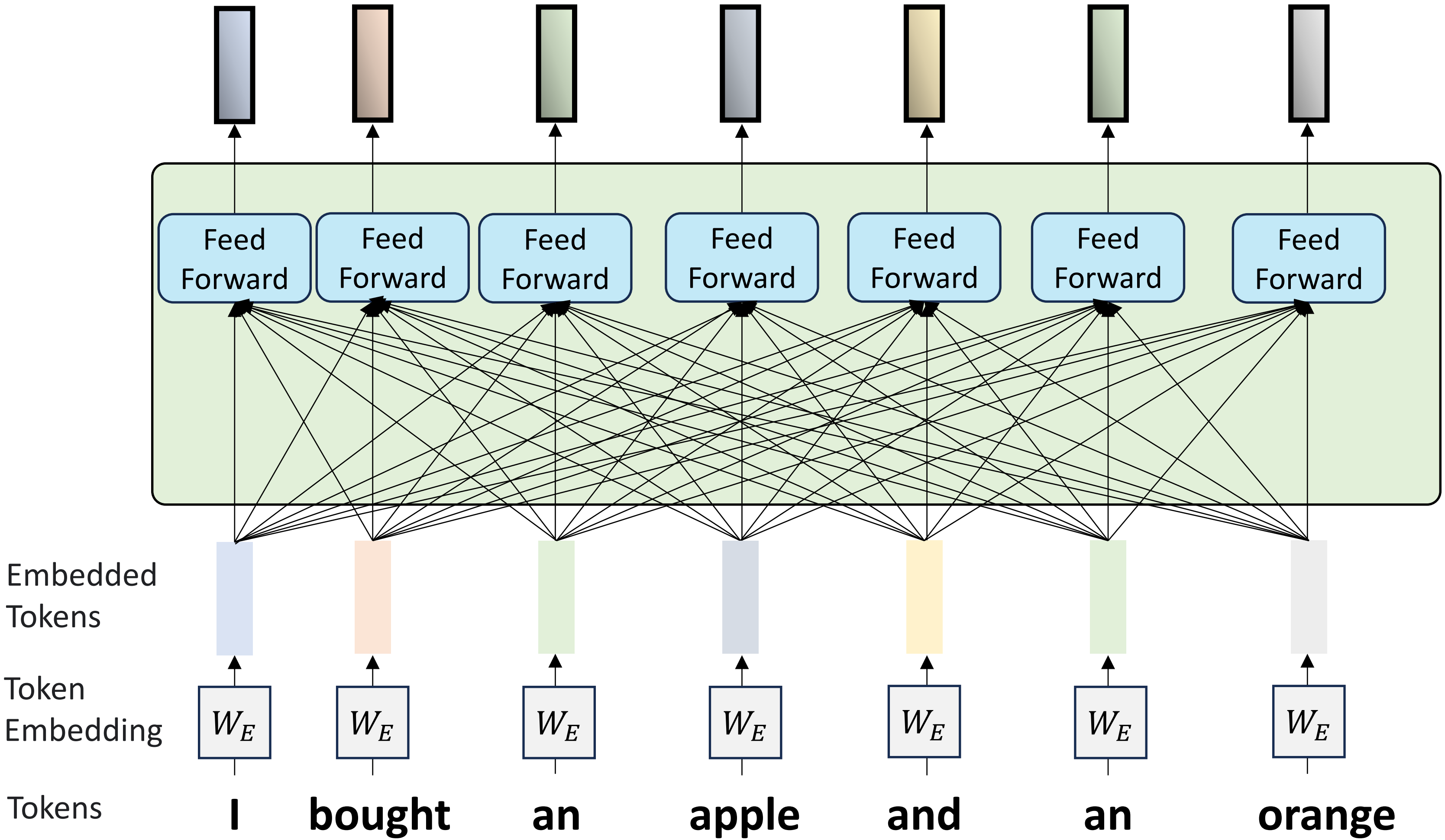


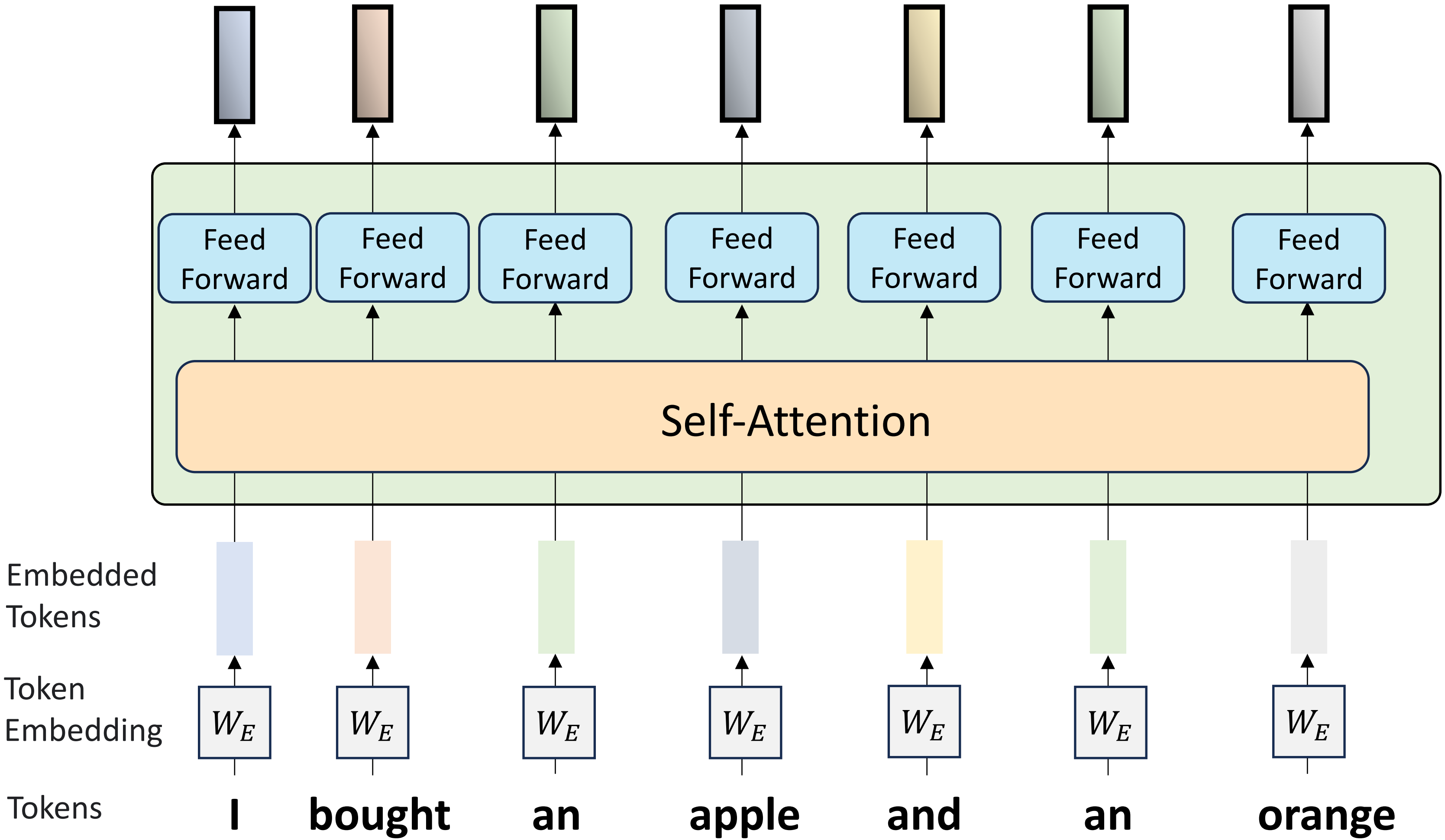



Encoder
Embedded Tokens
Token Embedding
W_E
Tokens
I
bought
an
apple
and
an
orange

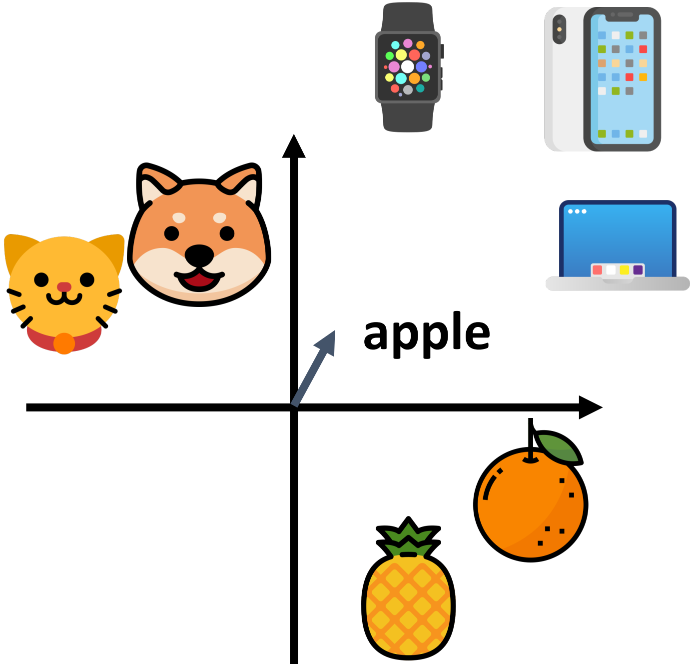

Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Embedded Tokens
Token Embedding
W_E
W_E
W_E
W_E
W_E
W_E
W_E
Tokens
I
bought
an
apple
and
an
orange


Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Embedded Tokens
Token Embedding
W_E
W_E
W_E
W_E
W_E
W_E
W_E
Tokens
I
bought
an
apple
and
an
orange


Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Embedded Tokens
Token Embedding
W_E
W_E
W_E
W_E
W_E
W_E
W_E
Tokens
I
bought
an
apple
and
an
orange


Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Self-Attention
Embedded Tokens
Token Embedding
W_E
W_E
W_E
W_E
W_E
W_E
W_E
Tokens
I
bought
an
apple
and
an
orange

Self-Attention

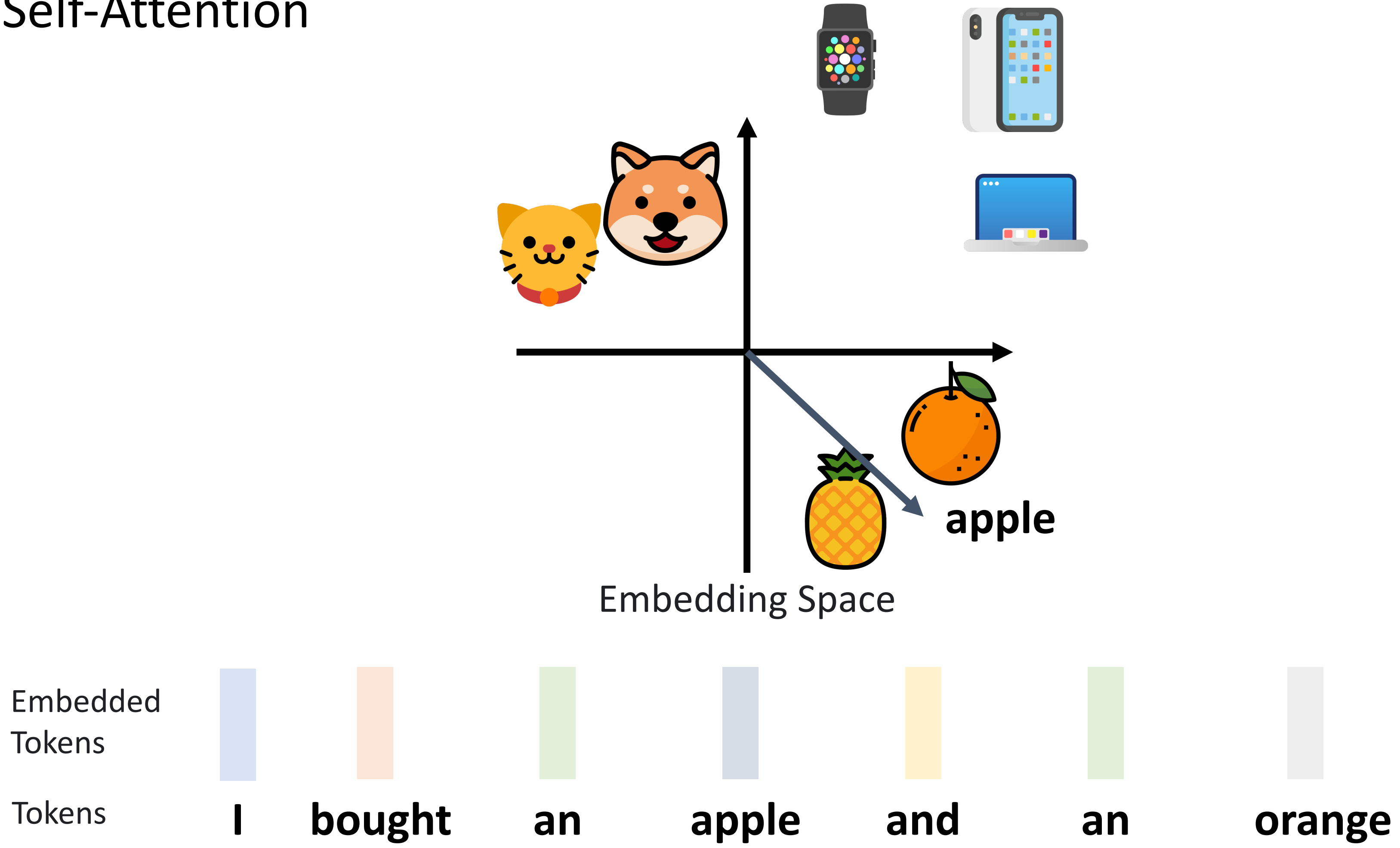


Embedding Space

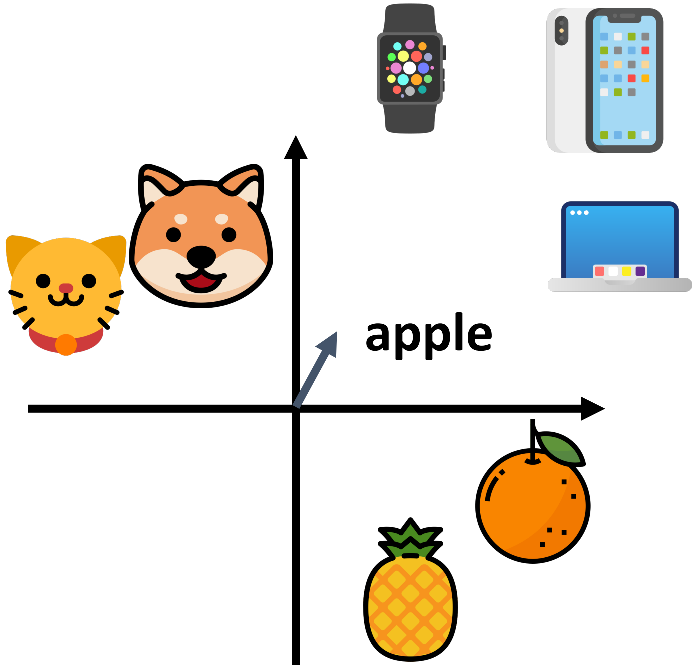
Embedded Tokens

Tokens I bought an apple and an orange

Self-Attention



Self-Attention

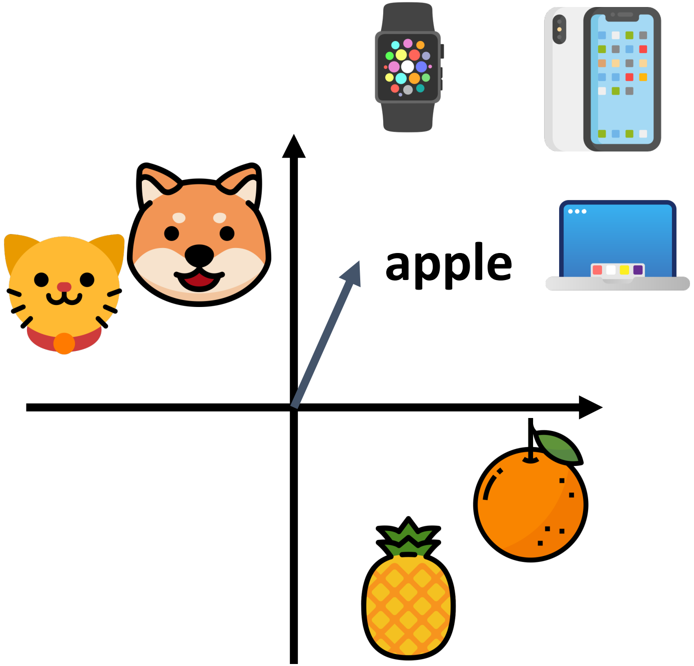


Embedding Space

Embedded Tokens

Tokens **I bought an apple watch**

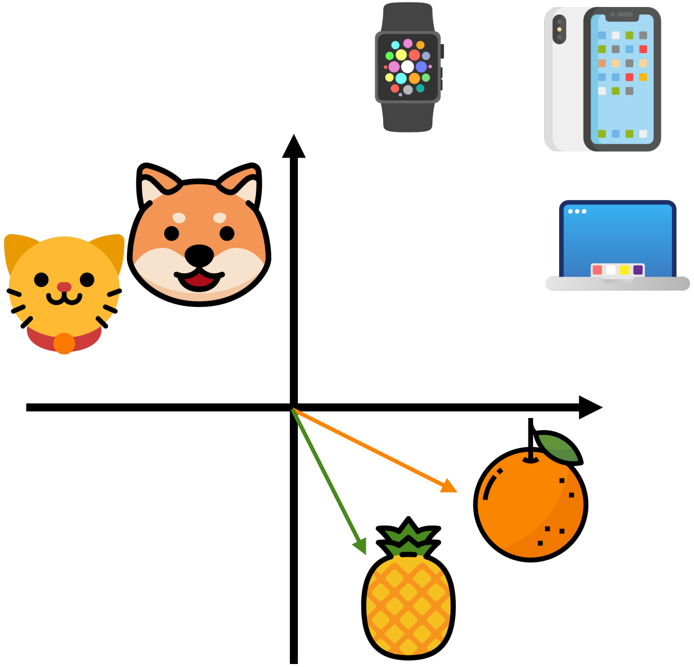
Self-Attention



Embedding Space

Embedded Tokens					
Tokens	**I**	**bought**	**an**	**apple**	**watch**

Self-Attention

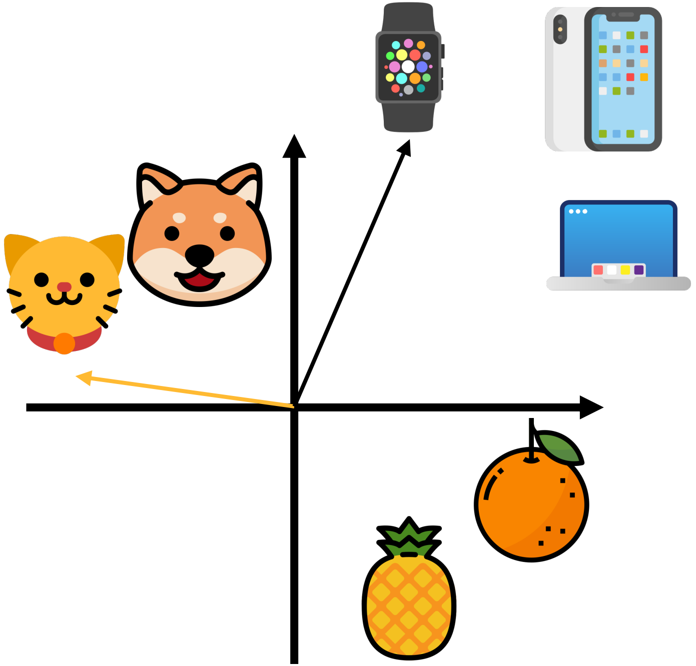


Embedding Space

Embedded Tokens

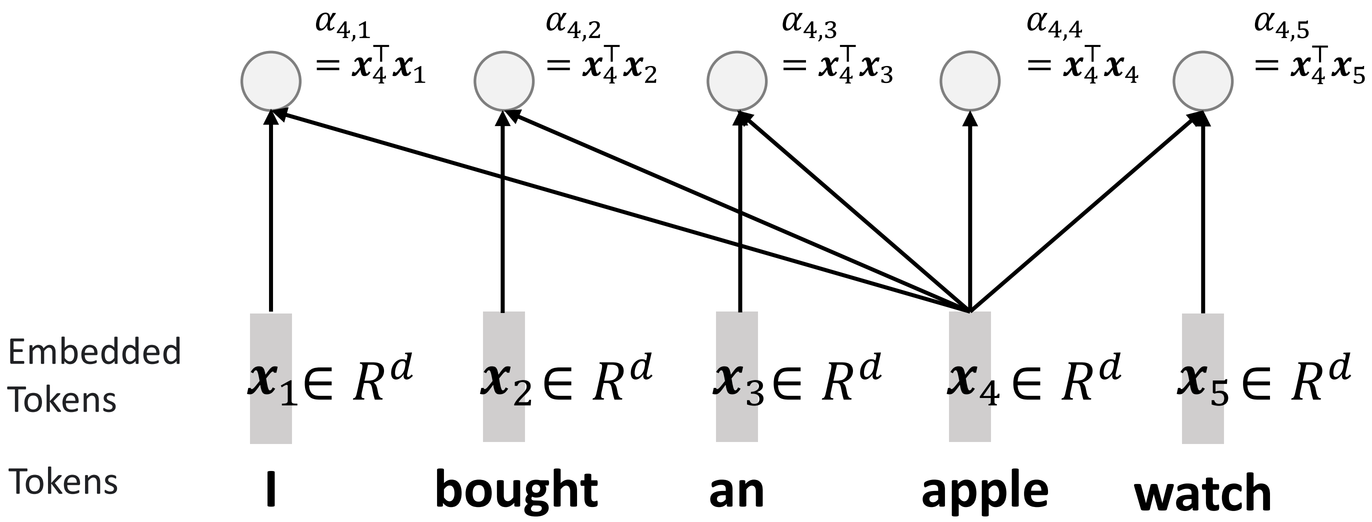
Tokens **I** **bought** **an** **apple** **watch**

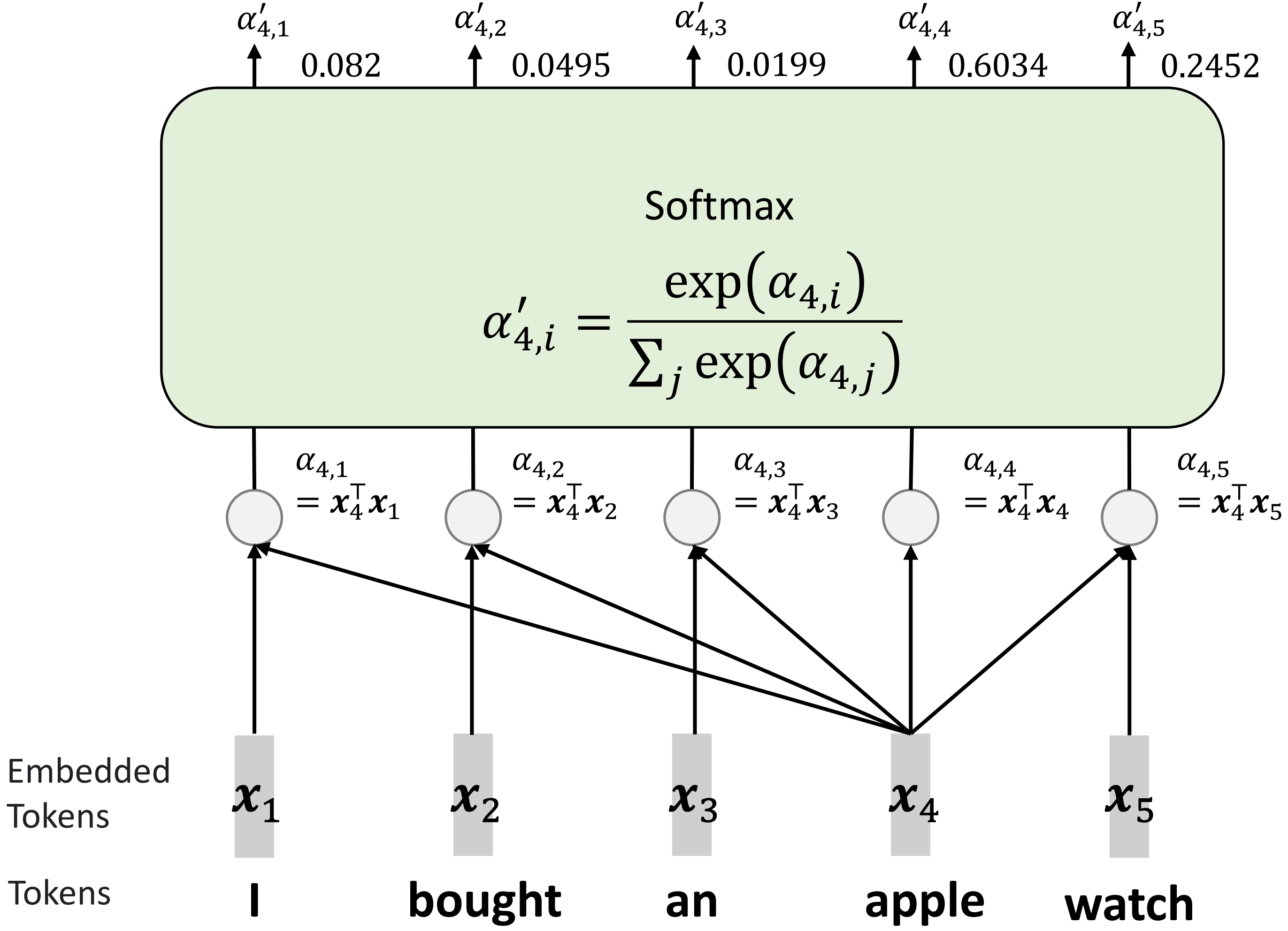
Self-Attention



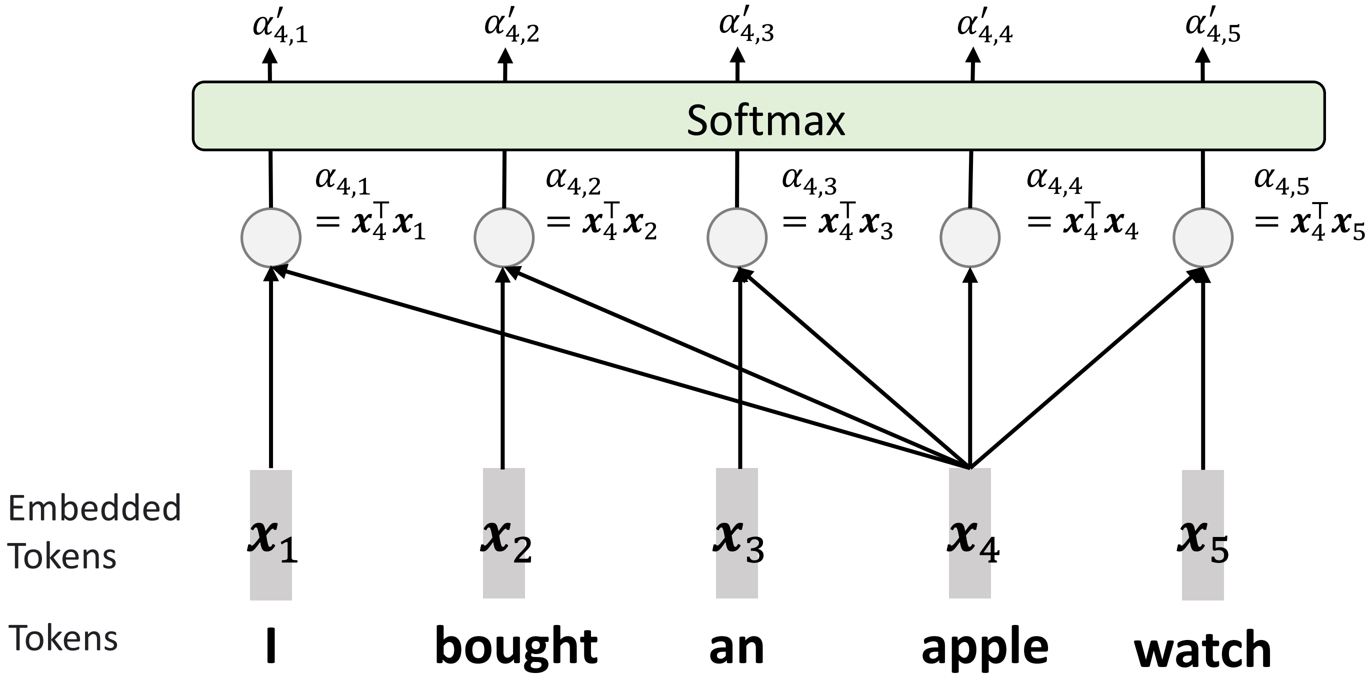
Embedding Space

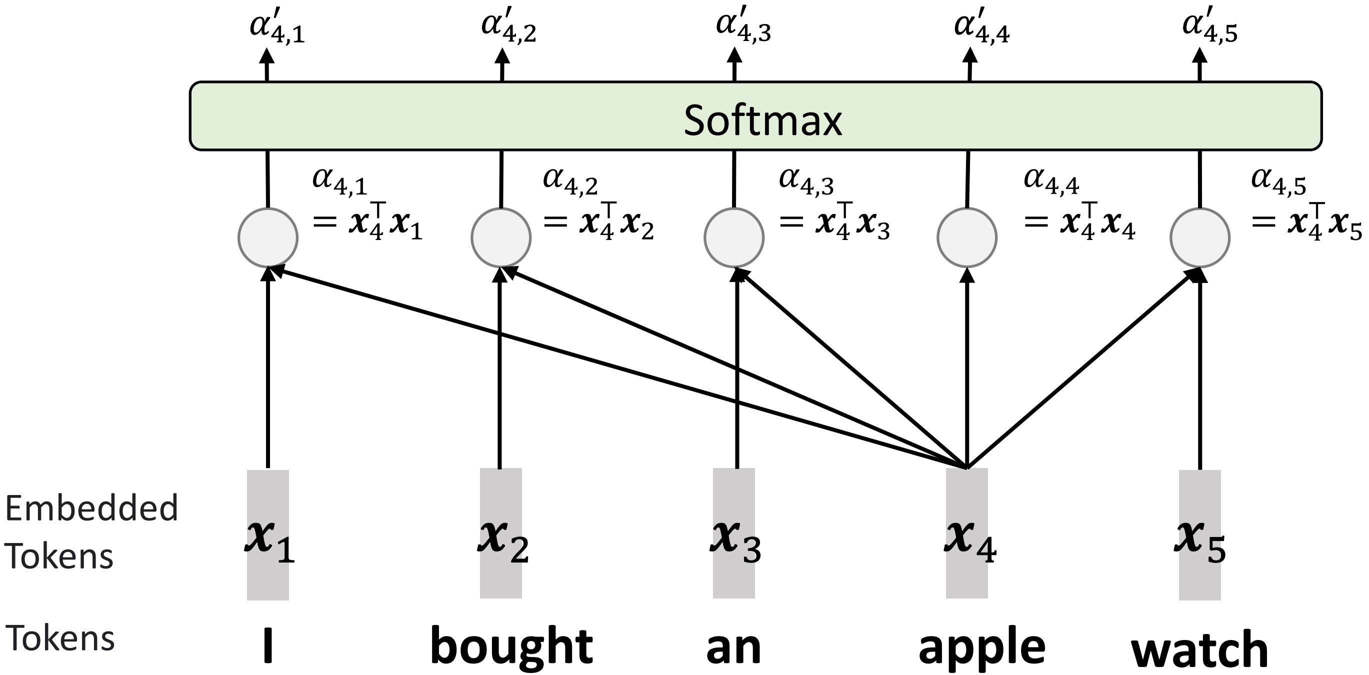
Self-Attention





$\alpha'_{4,1}$
0.082
$\alpha'_{4,2}$
0.0495
$\alpha'_{4,3}$
0.0199
$\alpha'_{4,4}$
0.6034
$\alpha'_{4,5}$
0.2452
Softmax
$\alpha'_{4,i} = \frac{\exp(\alpha_{4,i})}{\sum_j \exp(\alpha_{4,j})}$
$\alpha_{4,1} = \boldsymbol{x}_4^\top \boldsymbol{x}_1$
$\alpha_{4,2} = \boldsymbol{x}_4^\top \boldsymbol{x}_2$
$\alpha_{4,3} = \boldsymbol{x}_4^\top \boldsymbol{x}_3$
$\alpha_{4,4} = \boldsymbol{x}_4^\top \boldsymbol{x}_4$
$\alpha_{4,5} = \boldsymbol{x}_4^\top \boldsymbol{x}_5$
Embedded Tokens
$\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_3$ $\boldsymbol{x}_4$ $\boldsymbol{x}_5$
Tokens
I bought an apple watch

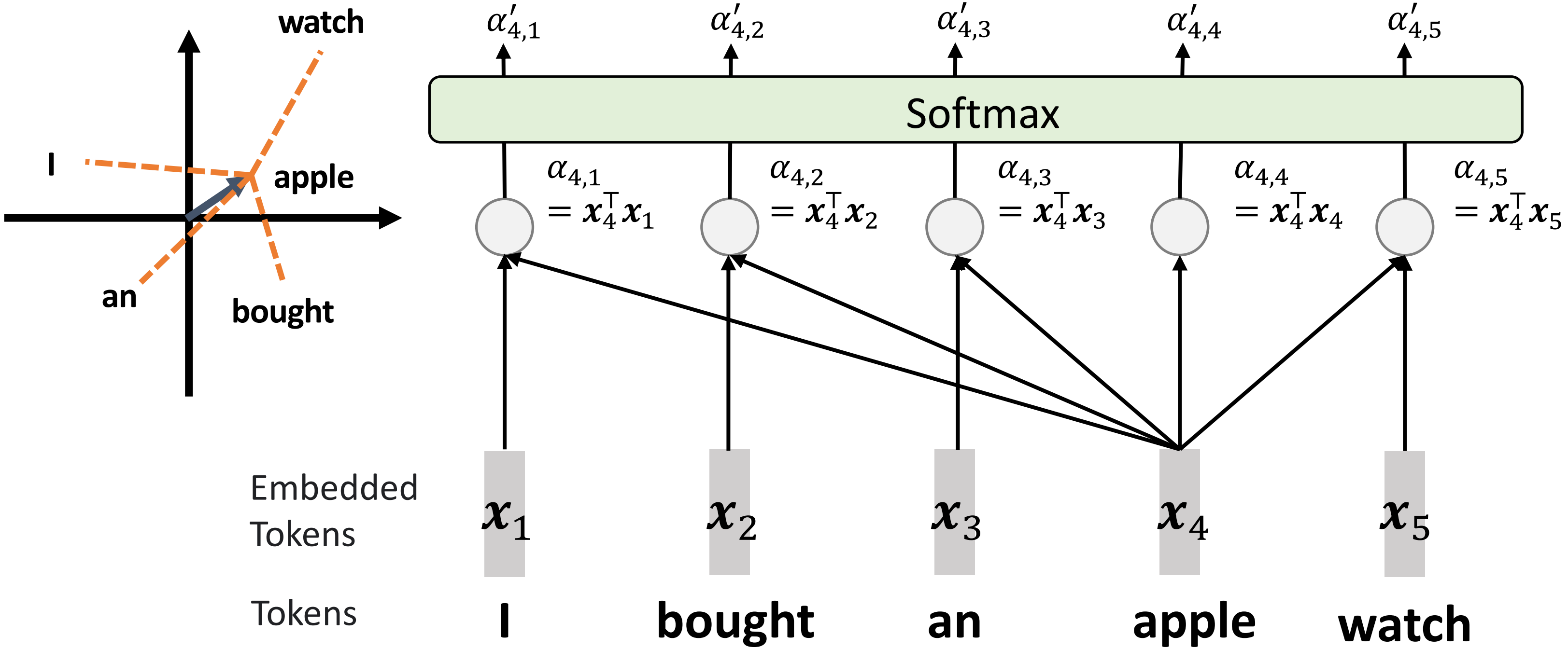

$\alpha'_{4,1}$
$\alpha'_{4,2}$
$\alpha'_{4,3}$
$\alpha'_{4,4}$
$\alpha'_{4,5}$
Softmax
$\alpha_{4,1} = \boldsymbol{x}_4^\top \boldsymbol{x}_1$
$\alpha_{4,2} = \boldsymbol{x}_4^\top \boldsymbol{x}_2$
$\alpha_{4,3} = \boldsymbol{x}_4^\top \boldsymbol{x}_3$
$\alpha_{4,4} = \boldsymbol{x}_4^\top \boldsymbol{x}_4$
$\alpha_{4,5} = \boldsymbol{x}_4^\top \boldsymbol{x}_5$
Embedded Tokens
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$
$\boldsymbol{x}_4$
$\boldsymbol{x}_5$
Tokens
I
bought
an
apple
watch



$\alpha'_{4,1}$
$\alpha'_{4,2}$
$\alpha'_{4,3}$
$\alpha'_{4,4}$
$\alpha'_{4,5}$
Softmax
$\alpha_{4,1} = \boldsymbol{x}_4^\top \boldsymbol{x}_1$
$\alpha_{4,2} = \boldsymbol{x}_4^\top \boldsymbol{x}_2$
$\alpha_{4,3} = \boldsymbol{x}_4^\top \boldsymbol{x}_3$
$\alpha_{4,4} = \boldsymbol{x}_4^\top \boldsymbol{x}_4$
$\alpha_{4,5} = \boldsymbol{x}_4^\top \boldsymbol{x}_5$
Embedded Tokens
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$
$\boldsymbol{x}_4$
$\boldsymbol{x}_5$
Tokens
I
bought
an
apple
watch

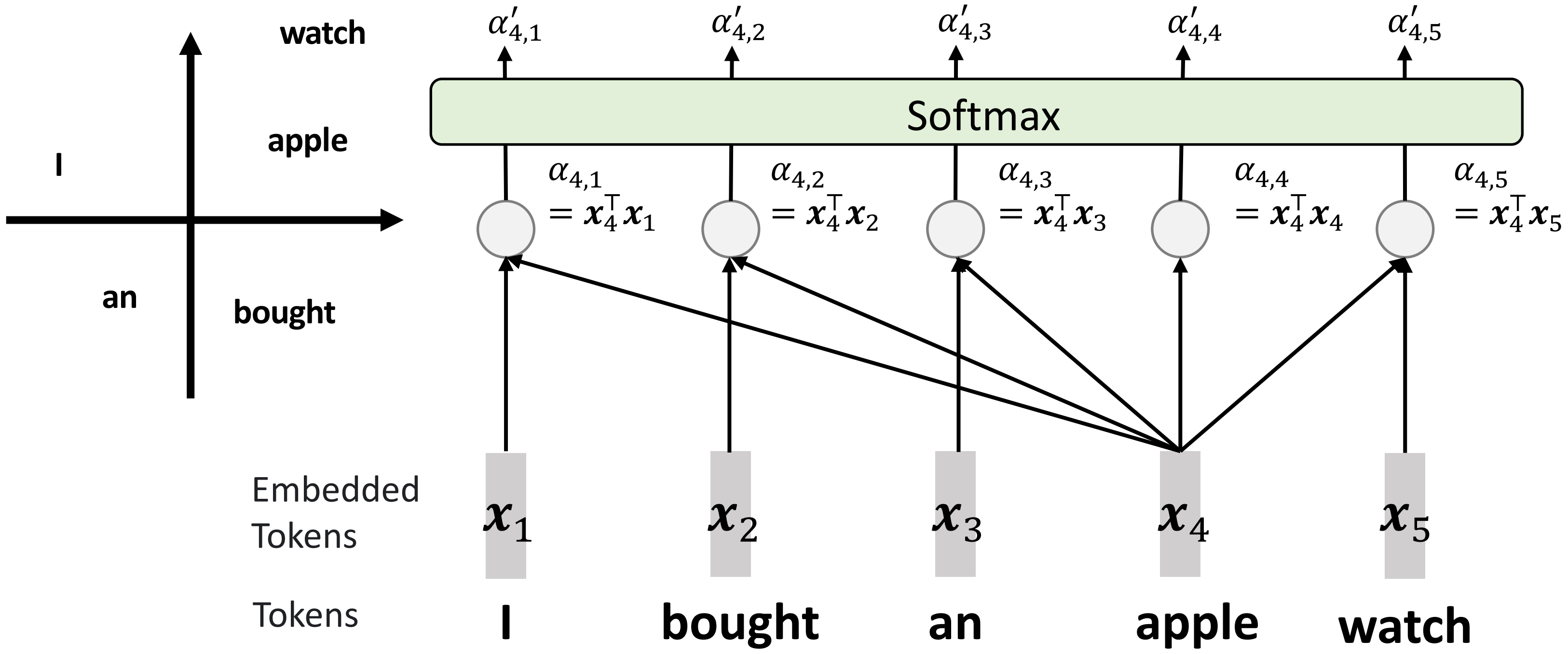
Updated feature

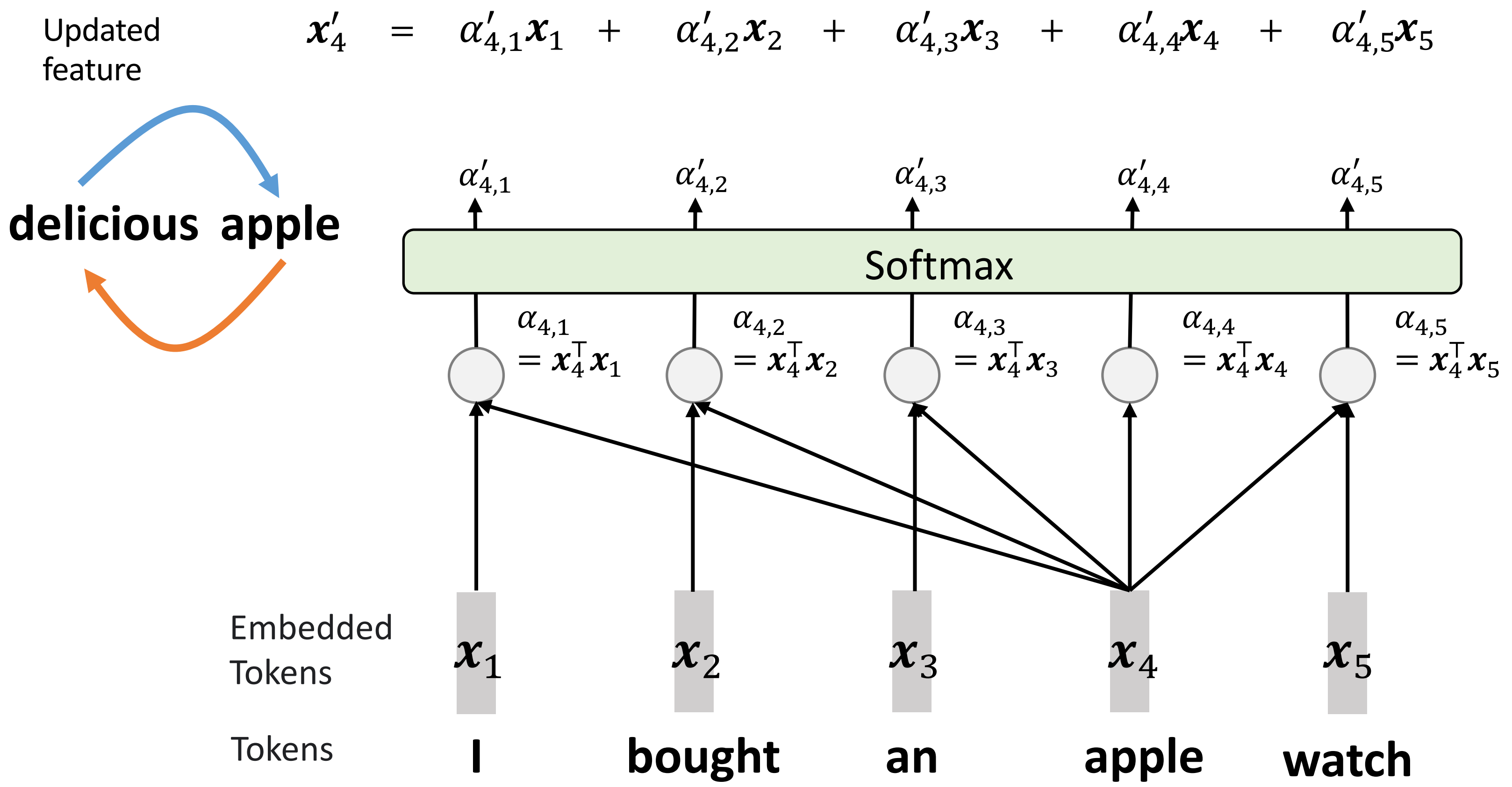
$$x'_4 = \alpha'_{4,1}x_1 + \alpha'_{4,2}x_2 + \alpha'_{4,3}x_3 + \alpha'_{4,4}x_4 + \alpha'_{4,5}x_5$$

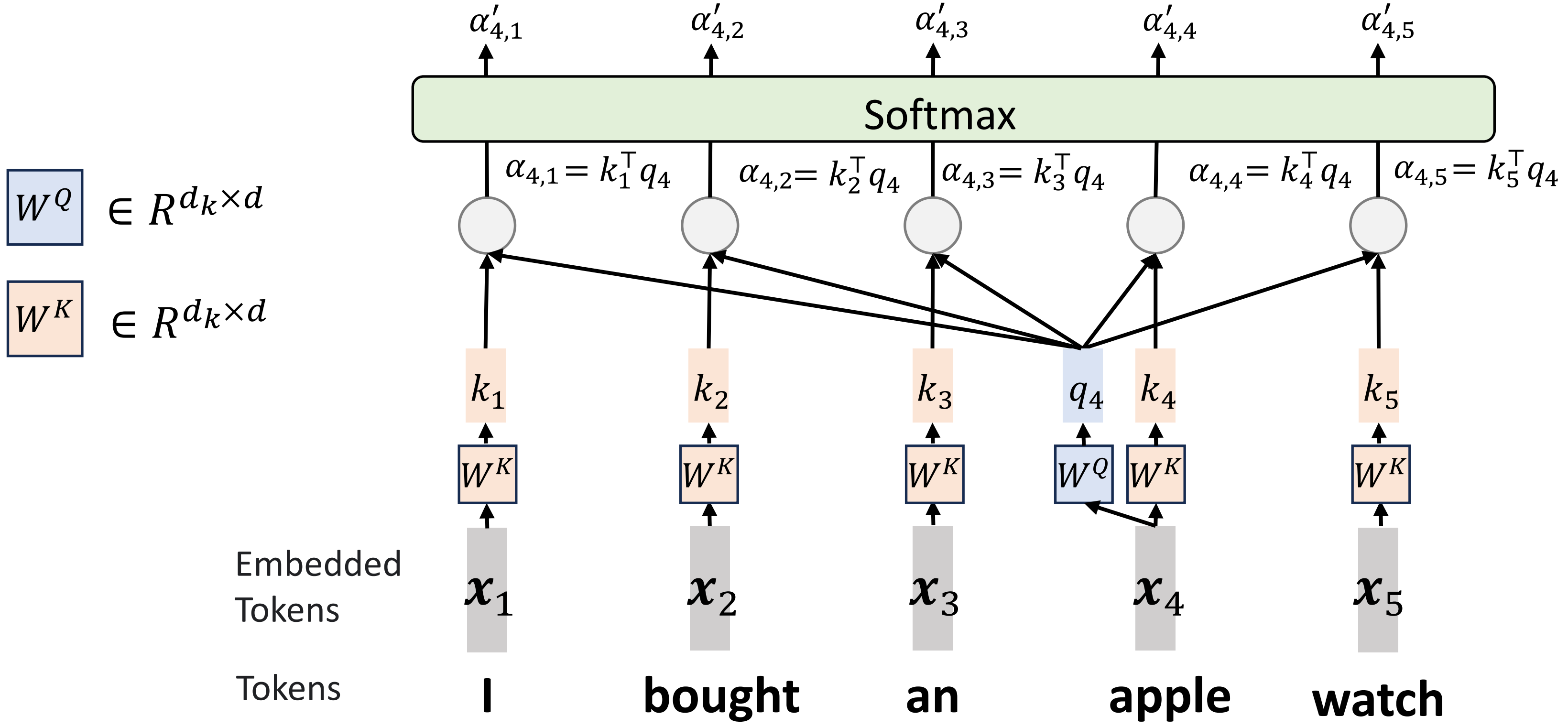


Updated feature

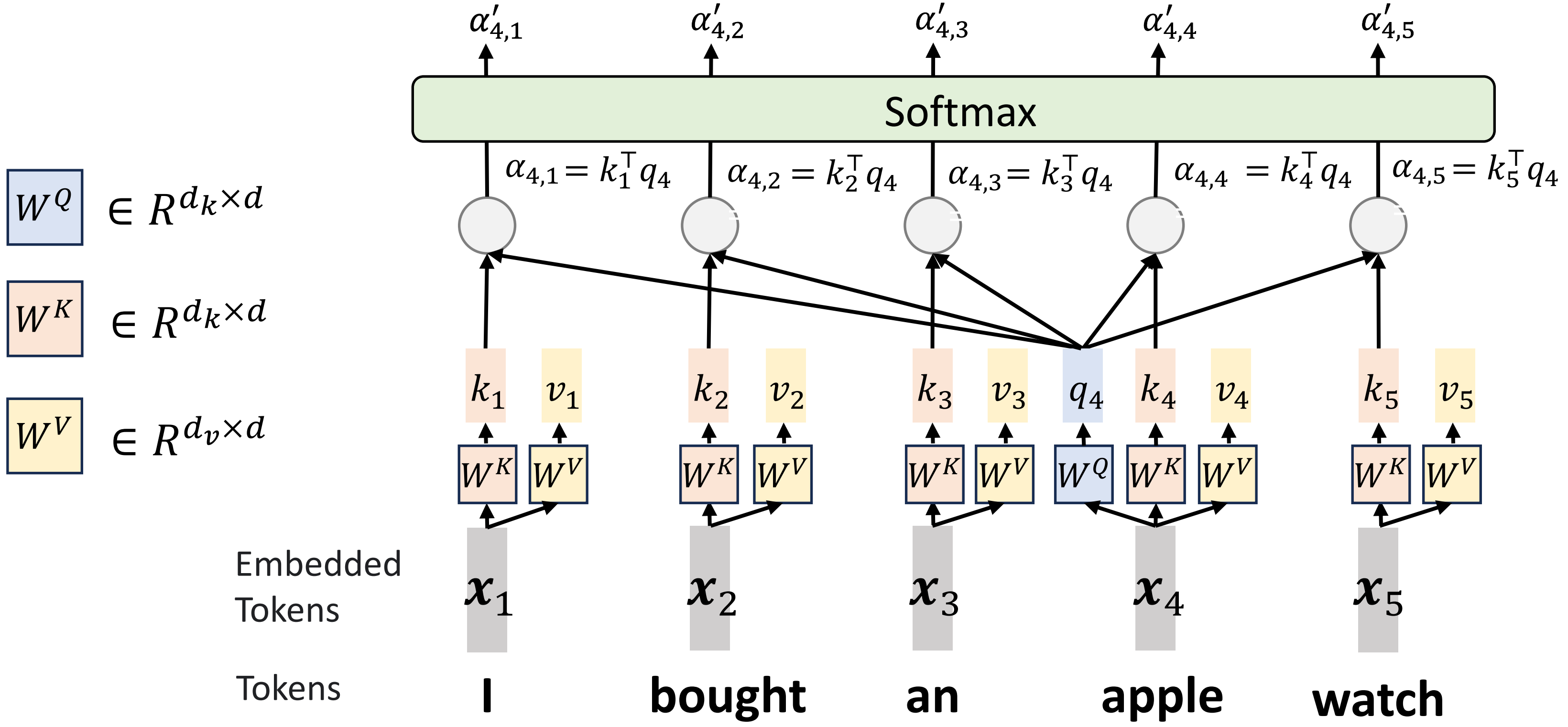
$$\boldsymbol{x}'_4 = \alpha'_{4,1}\boldsymbol{x}_1 + \alpha'_{4,2}\boldsymbol{x}_2 + \alpha'_{4,3}\boldsymbol{x}_3 + \alpha'_{4,4}\boldsymbol{x}_4 + \alpha'_{4,5}\boldsymbol{x}_5$$




Updated feature
$x'_4 = \alpha'_{4,1}x_1 + \alpha'_{4,2}x_2 + \alpha'_{4,3}x_3 + \alpha'_{4,4}x_4 + \alpha'_{4,5}x_5$
delicious apple
$\alpha'_{4,1}$
$\alpha'_{4,2}$
$\alpha'_{4,3}$
$\alpha'_{4,4}$
$\alpha'_{4,5}$
Softmax
$\alpha_{4,1} = x_4^\top x_1$
$\alpha_{4,2} = x_4^\top x_2$
$\alpha_{4,3} = x_4^\top x_3$
$\alpha_{4,4} = x_4^\top x_4$
$\alpha_{4,5} = x_4^\top x_5$
Embedded Tokens
x_1
x_2
x_3
x_4
x_5
Tokens
I
bought
an
apple
watch



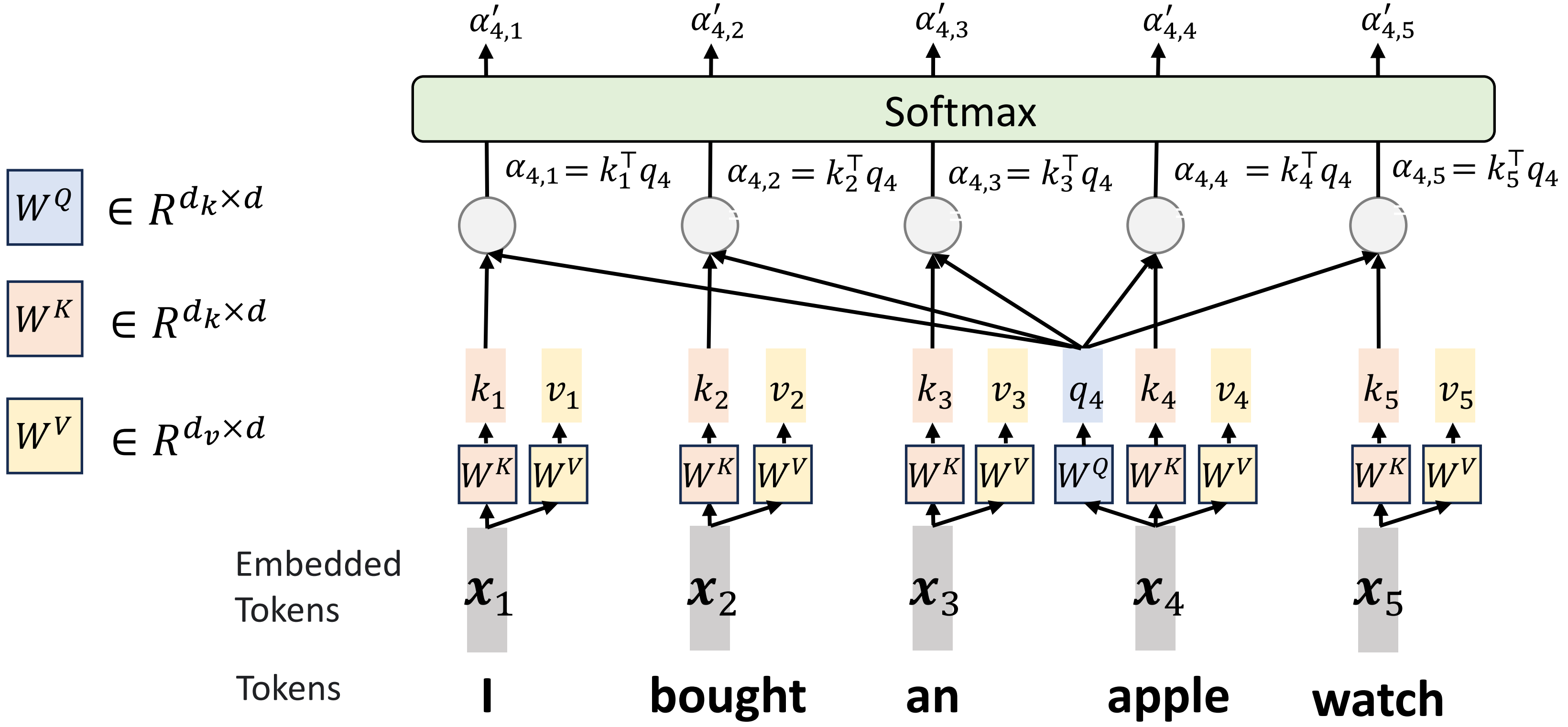
$\alpha'_{4,1}$
$\alpha'_{4,2}$
$\alpha'_{4,3}$
$\alpha'_{4,4}$
$\alpha'_{4,5}$
Softmax
$\alpha_{4,1} = k_1^\top q_4$
$\alpha_{4,2} = k_2^\top q_4$
$\alpha_{4,3} = k_3^\top q_4$
$\alpha_{4,4} = k_4^\top q_4$
$\alpha_{4,5} = k_5^\top q_4$
$W^Q \in R^{d_k \times d}$
$W^K \in R^{d_k \times d}$
k_1
k_2
k_3
q_4
k_4
k_5
W^K
W^K
W^K
W^Q
W^K
W^K
Embedded Tokens
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$
$\boldsymbol{x}_4$
$\boldsymbol{x}_5$
Tokens
I
bought
an
apple
watch



$\alpha'_{4,1}$
$\alpha'_{4,2}$
$\alpha'_{4,3}$
$\alpha'_{4,4}$
$\alpha'_{4,5}$
Softmax
$\alpha_{4,1} = k_1^\top q_4$
$\alpha_{4,2} = k_2^\top q_4$
$\alpha_{4,3} = k_3^\top q_4$
$\alpha_{4,4} = k_4^\top q_4$
$\alpha_{4,5} = k_5^\top q_4$
$W^Q \in R^{d_k \times d}$
$W^K \in R^{d_k \times d}$
$W^V \in R^{d_v \times d}$
k_1 v_1 k_2 v_2 k_3 v_3 q_4 k_4 v_4 k_5 v_5
W^K W^V W^K W^V W^K W^V W^Q W^K W^V W^K W^V
Embedded Tokens
$\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_3$ $\boldsymbol{x}_4$ $\boldsymbol{x}_5$
Tokens
I
bought
an
apple
watch

Updated feature

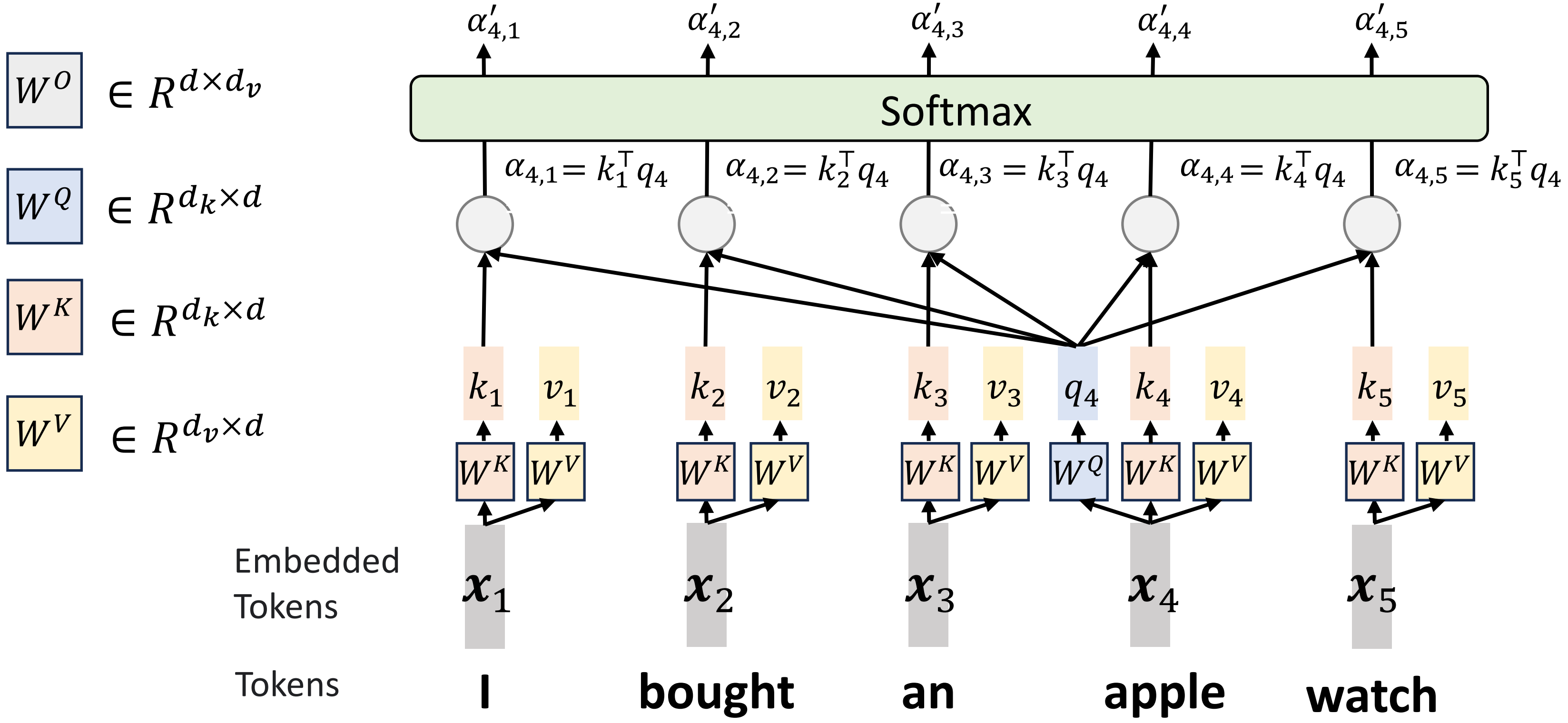
$$\boldsymbol{x}'_4 = \alpha'_{4,1} v_1 + \alpha'_{4,2} v_2 + \alpha'_{4,3} v_3 + \alpha'_{4,4} v_4 + \alpha'_{4,5} v_5$$

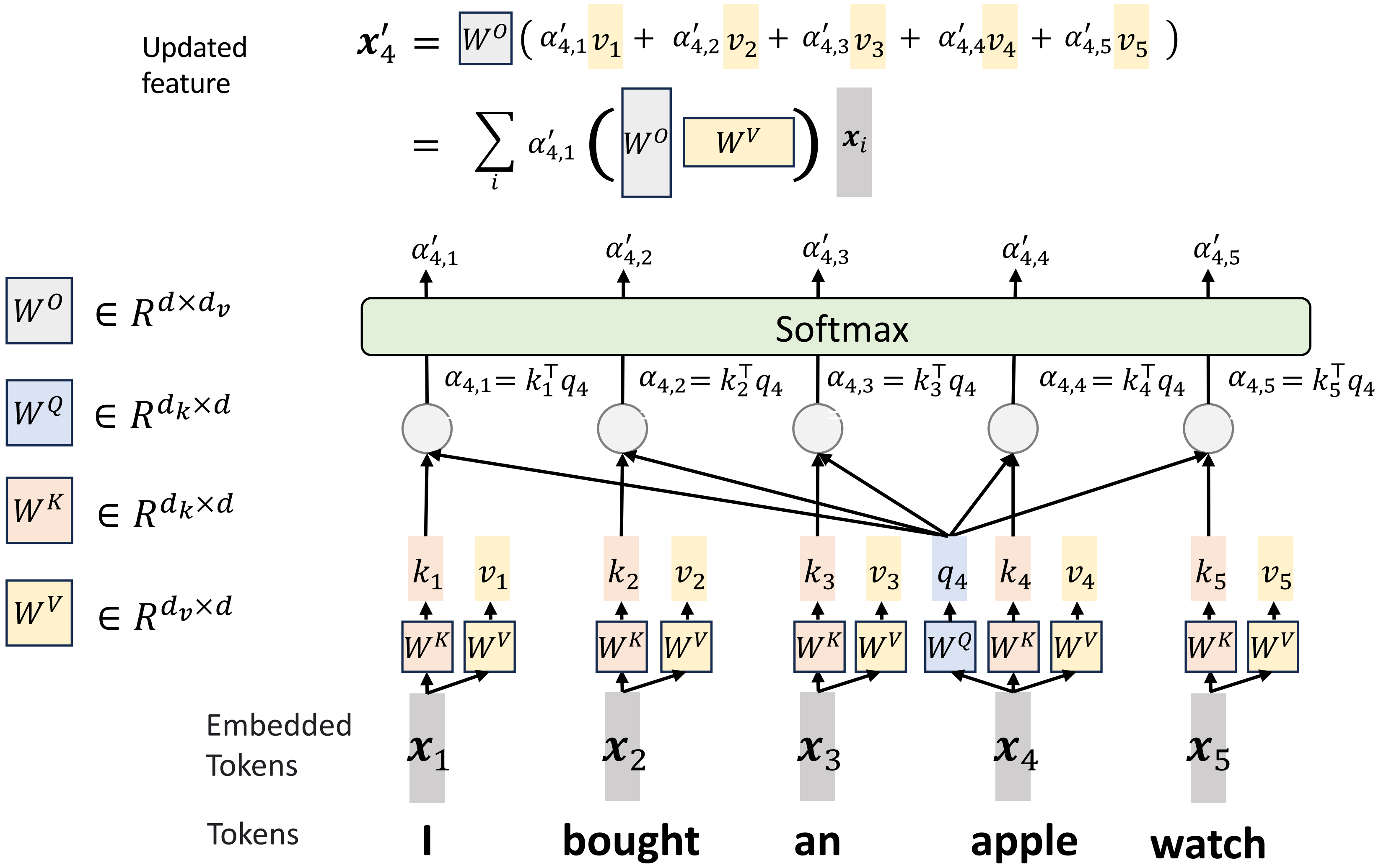


Updated feature

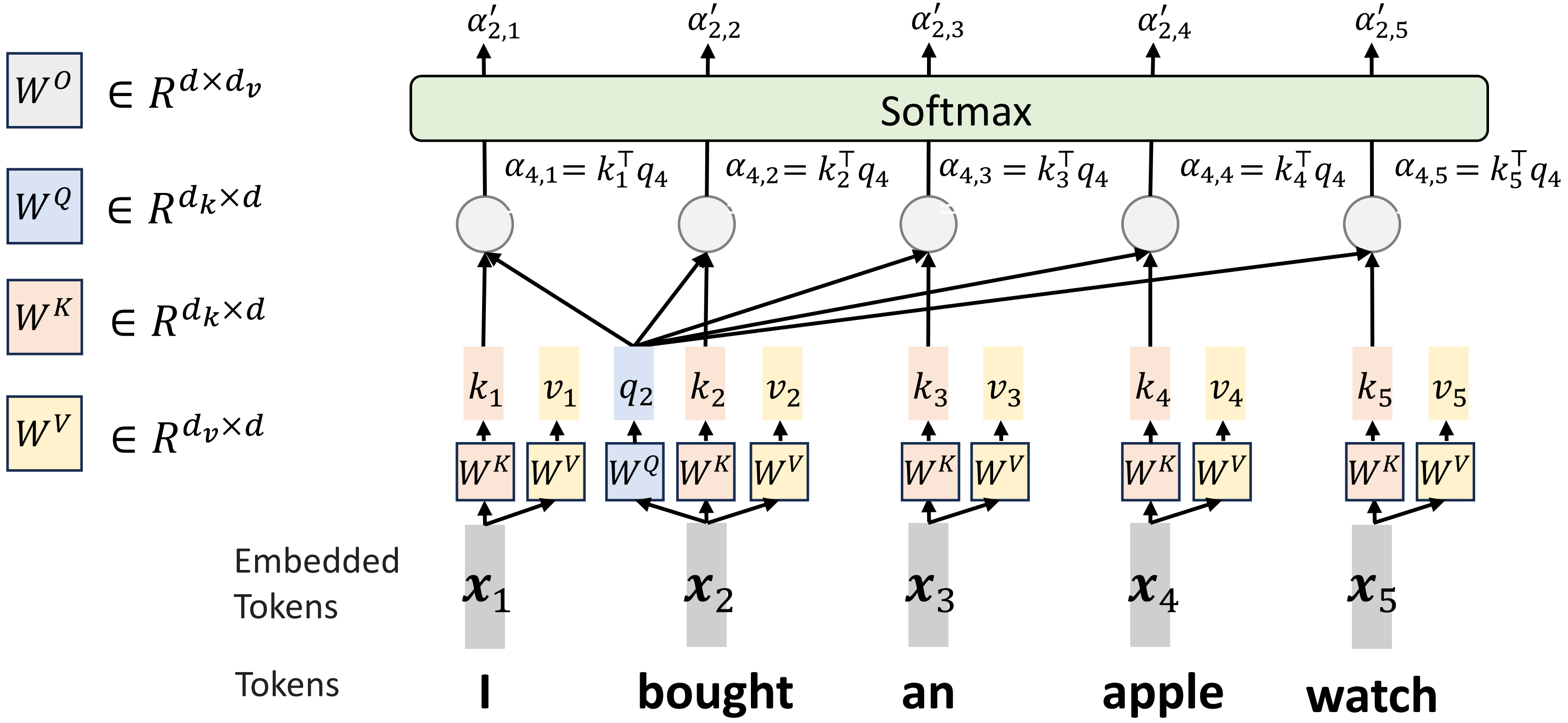
$$\boldsymbol{x}'_4 = \boxed{W^O}\left(\alpha'_{4,1}v_1 + \alpha'_{4,2}v_2 + \alpha'_{4,3}v_3 + \alpha'_{4,4}v_4 + \alpha'_{4,5}v_5\right)$$

$$= \sum_i \alpha'_{4,1}\, \boxed{W^O}\, \boxed{W^V}\, \boldsymbol{x}_i$$

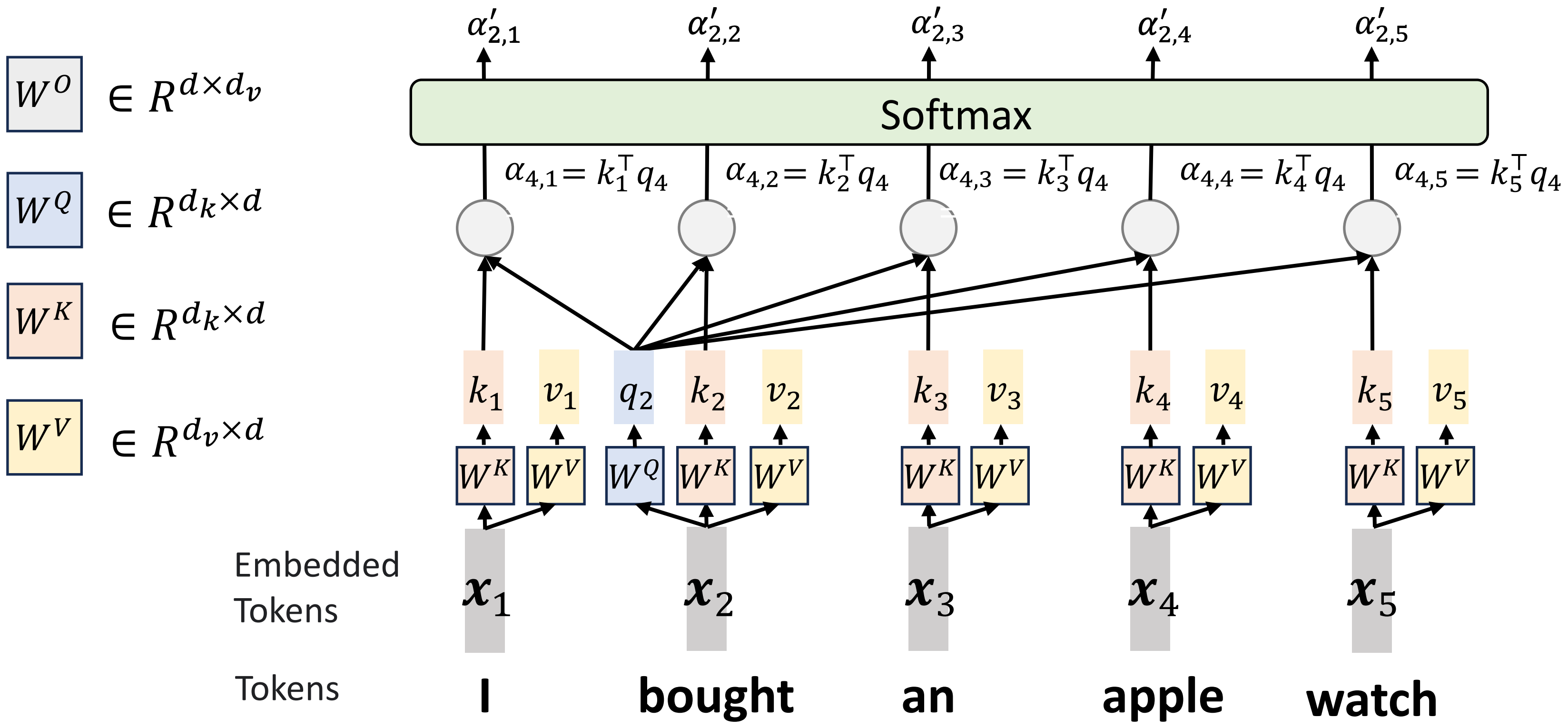




Updated feature
$x'_4 = W^O (\alpha'_{4,1} v_1 + \alpha'_{4,2} v_2 + \alpha'_{4,3} v_3 + \alpha'_{4,4} v_4 + \alpha'_{4,5} v_5)$
$= \sum_i \alpha'_{4,1} (W^O W^V) x_i$
$W^O \in R^{d \times d_v}$
$W^Q \in R^{d_k \times d}$
$W^K \in R^{d_k \times d}$
$W^V \in R^{d_v \times d}$
$\alpha'_{4,1}$ $\alpha'_{4,2}$ $\alpha'_{4,3}$ $\alpha'_{4,4}$ $\alpha'_{4,5}$
Softmax
$\alpha_{4,1} = k_1^\top q_4$
$\alpha_{4,2} = k_2^\top q_4$
$\alpha_{4,3} = k_3^\top q_4$
$\alpha_{4,4} = k_4^\top q_4$
$\alpha_{4,5} = k_5^\top q_4$
k_1 v_1 k_2 v_2 k_3 v_3 q_4 k_4 v_4 k_5 v_5
W^K W^V W^Q
Embedded Tokens
x_1 x_2 x_3 x_4 x_5
Tokens
I
bought
an
apple
watch

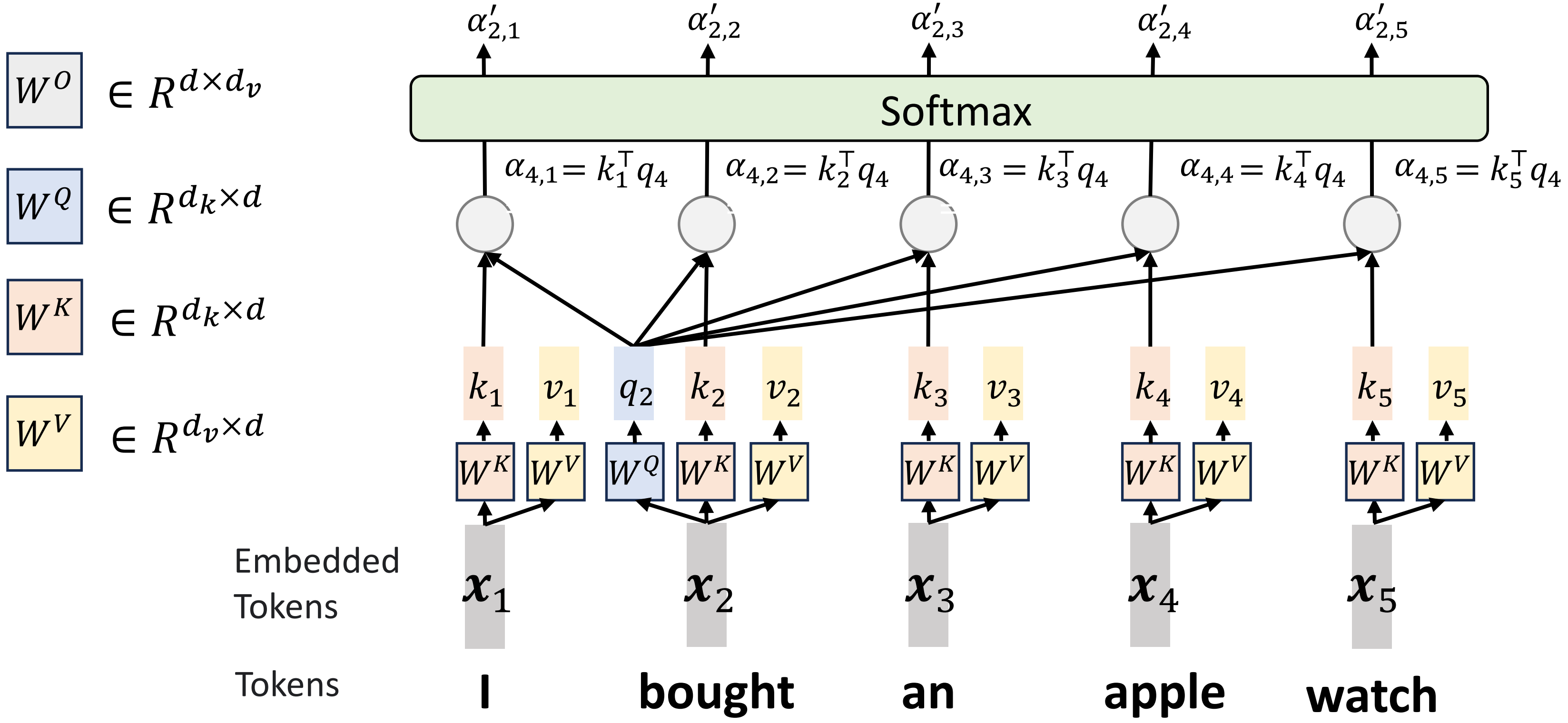


$W^O \in R^{d \times d_v}$
$W^Q \in R^{d_k \times d}$
$W^K \in R^{d_k \times d}$
$W^V \in R^{d_v \times d}$
$\alpha'_{2,1}$
$\alpha'_{2,2}$
$\alpha'_{2,3}$
$\alpha'_{2,4}$
$\alpha'_{2,5}$
Softmax
$\alpha_{4,1} = k_1^\top q_4$
$\alpha_{4,2} = k_2^\top q_4$
$\alpha_{4,3} = k_3^\top q_4$
$\alpha_{4,4} = k_4^\top q_4$
$\alpha_{4,5} = k_5^\top q_4$
k_1 v_1 q_2 k_2 v_2 k_3 v_3 k_4 v_4 k_5 v_5
W^K W^V W^Q W^K W^V W^K W^V W^K W^V W^K W^V
Embedded Tokens
$\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_3$ $\boldsymbol{x}_4$ $\boldsymbol{x}_5$
Tokens
I bought an apple watch



$W^O \in R^{d \times d_v}$
$W^Q \in R^{d_k \times d}$
$W^K \in R^{d_k \times d}$
$W^V \in R^{d_v \times d}$
$\alpha'_{2,1}$
$\alpha'_{2,2}$
$\alpha'_{2,3}$
$\alpha'_{2,4}$
$\alpha'_{2,5}$
Softmax
$\alpha_{4,1} = k_1^\top q_4$
$\alpha_{4,2} = k_2^\top q_4$
$\alpha_{4,3} = k_3^\top q_4$
$\alpha_{4,4} = k_4^\top q_4$
$\alpha_{4,5} = k_5^\top q_4$
k_1 v_1 q_2 k_2 v_2 k_3 v_3 k_4 v_4 k_5 v_5
W^K W^V W^Q W^K W^V W^K W^V W^K W^V W^K W^V
Embedded Tokens
$\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_3$ $\boldsymbol{x}_4$ $\boldsymbol{x}_5$
Tokens
I
bought
an
apple
watch

Updated feature $$\boldsymbol{x}_2' = W^O\left(\alpha_{2,1}' v_1 + \alpha_{2,2}' v_2 + \alpha_{2,3}' v_3 + \alpha_{2,4}' v_4 + \alpha_{2,5}' v_5\right)$$



Total weights: 175,181,291,520

Organized into 27,938 matrices

GPT-3

Embedding	$\overset{12,288}{d\_embed} * \overset{50,257}{n\_vocab}$	= 617,558,016
Key	$\overset{128}{d\_query} * \overset{12,288}{d\_embed} * \overset{96}{n\_heads} * \overset{96}{n\_layers}$	= 14,495,514,624
Query	$\overset{128}{d\_query} * \overset{12,288}{d\_embed} * \overset{96}{n\_heads} * \overset{96}{n\_layers}$	= 14,495,514,624
Value	$\overset{128}{d\_value} * \overset{12,288}{d\_embed} * \overset{96}{n\_heads} * \overset{96}{n\_layers}$	= 14,495,514,624
Output	$\overset{12,288}{d\_embed} * \overset{128}{d\_value} * \overset{96}{n\_heads} * \overset{96}{n\_layers}$	= 14,495,514,624
Up-projection	$\overset{49,152}{n\_neurons} * \overset{12,288}{d\_embed} * \overset{96}{n\_layers}$	= 57,982,058,496
Down-projection	$\overset{12,288}{d\_embed} * \overset{49,152}{n\_neurons} * \overset{96}{n\_layers}$	= 57,982,058,496
Unembedding	$\overset{50,257}{n\_vocab} * \overset{12,288}{d\_embed}$	= 617,558,016

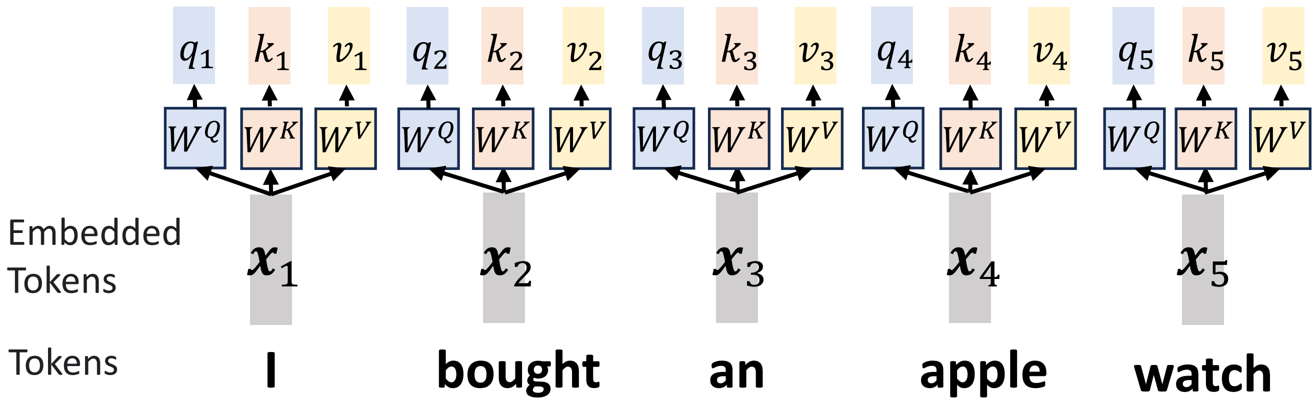
$$\alpha_{1,1} = k_1^\top q_1$$

$$\alpha_{1,2} = k_2^\top q_1$$

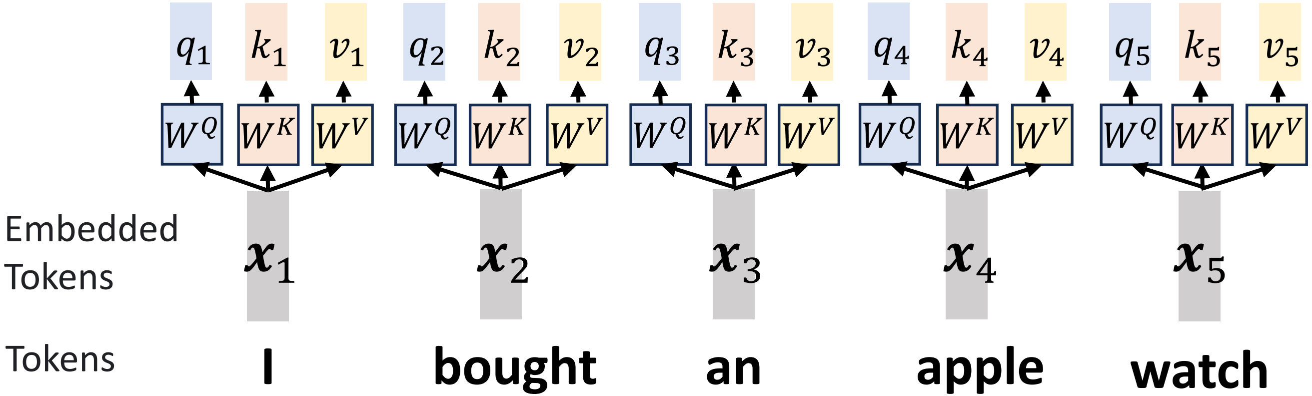
$$\alpha_{1,3} = k_3^\top q_1$$

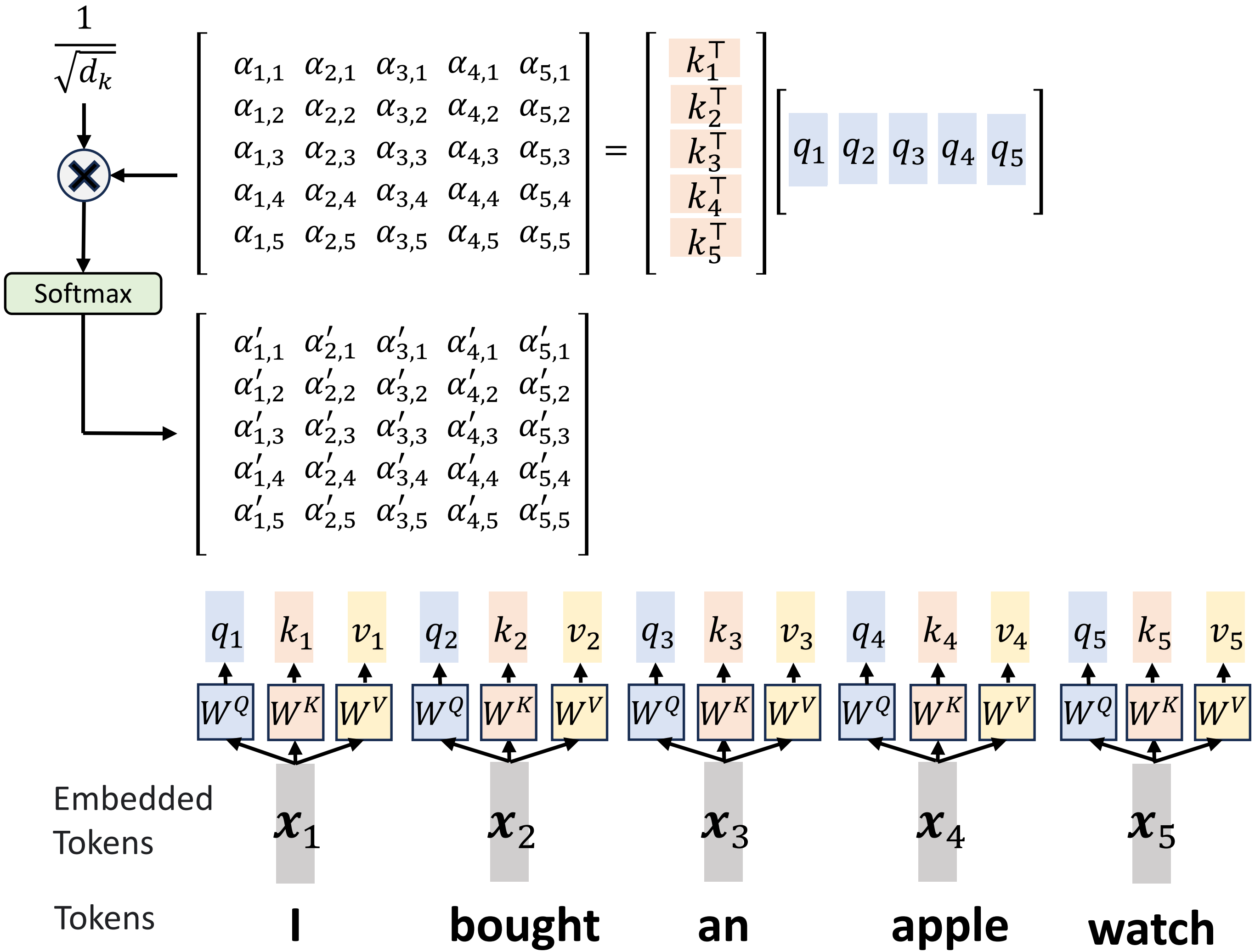
$$\alpha_{1,4} = k_4^\top q_1$$

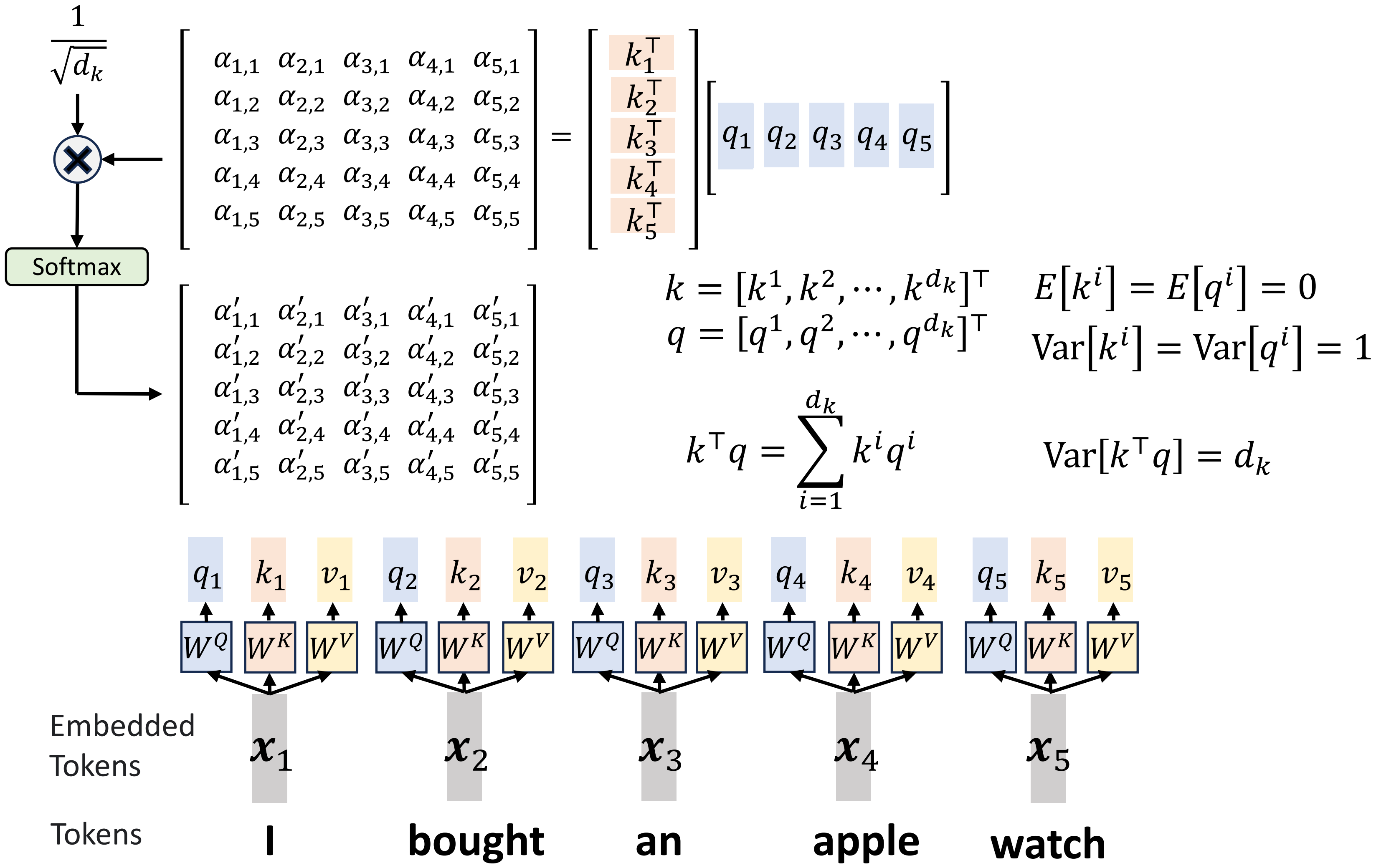
$$\alpha_{1,5} = k_5^\top q_1$$

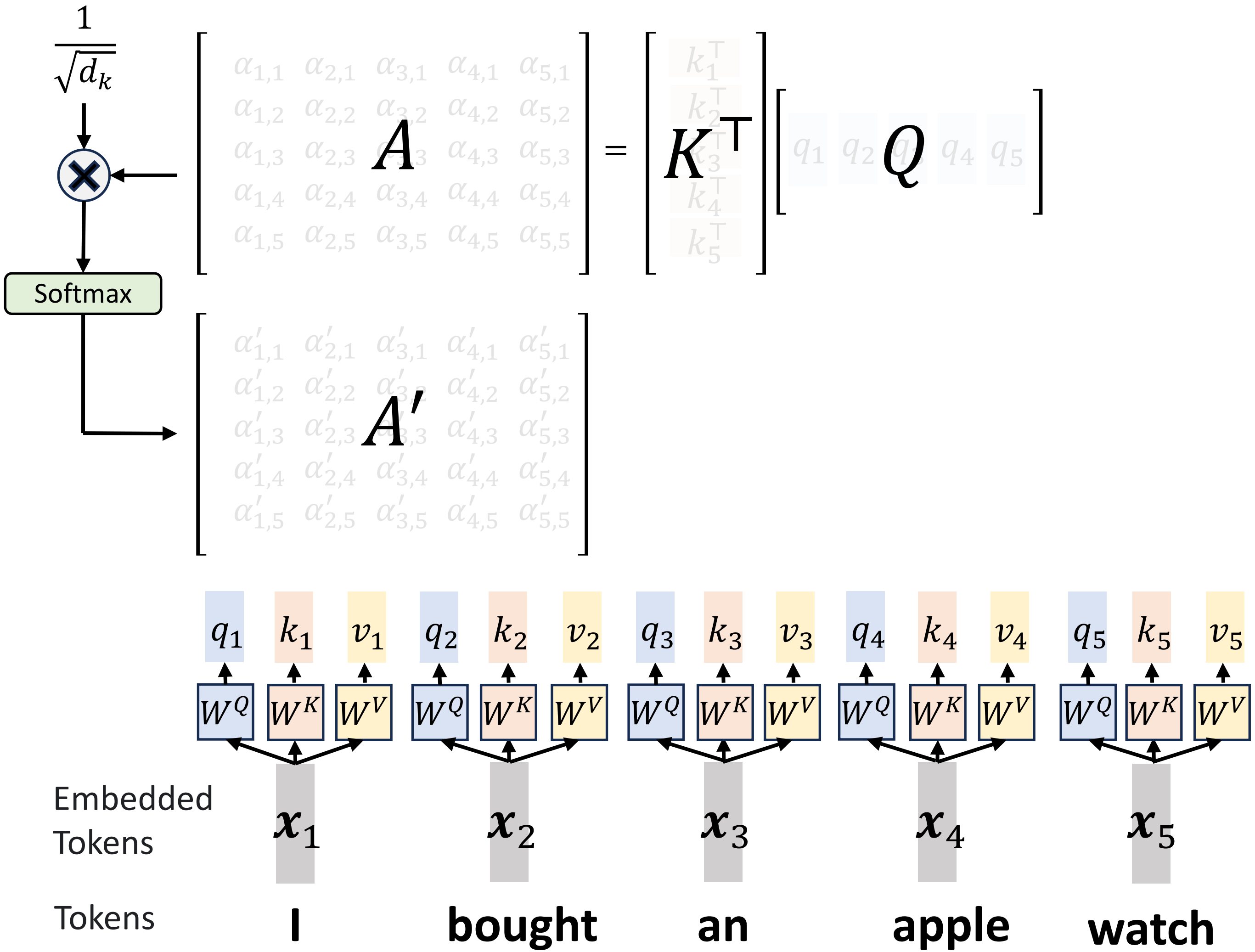


$$
\begin{matrix}
\alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1} & \alpha_{5,1} \\
\alpha_{1,2} & \alpha_{2,2} & \alpha_{3,2} & \alpha_{4,2} & \alpha_{5,2} \\
\alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3} & \alpha_{5,3} \\
\alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4} & \alpha_{5,4} \\
\alpha_{1,5} & \alpha_{2,5} & \alpha_{3,5} & \alpha_{4,5} & \alpha_{5,5}
\end{matrix}
=
\begin{bmatrix}
k_1^\top \\ k_2^\top \\ k_3^\top \\ k_4^\top \\ k_5^\top
\end{bmatrix}
\begin{matrix} q_1 & q_2 & q_3 & q_4 & q_5 \end{matrix}
$$

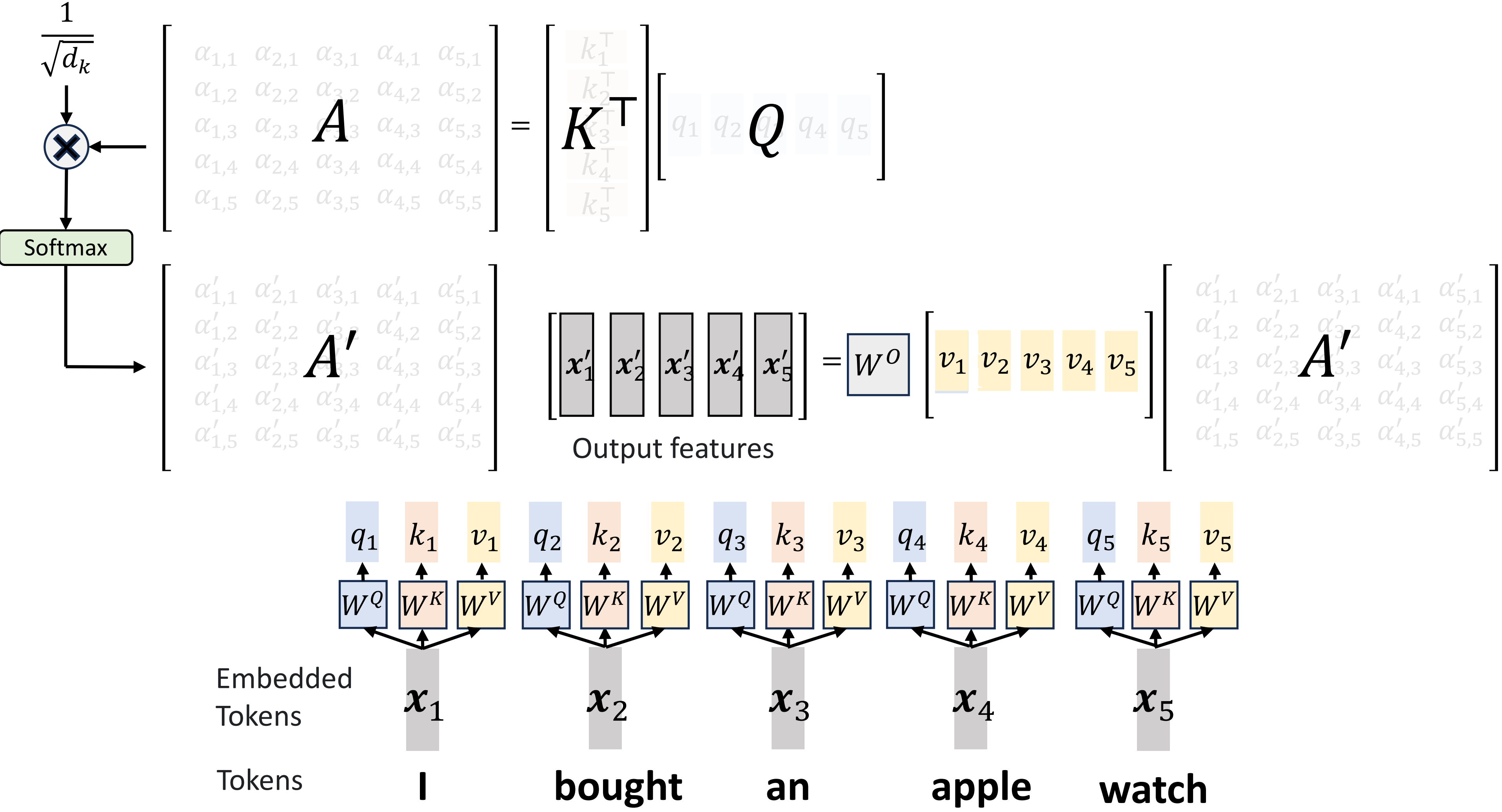


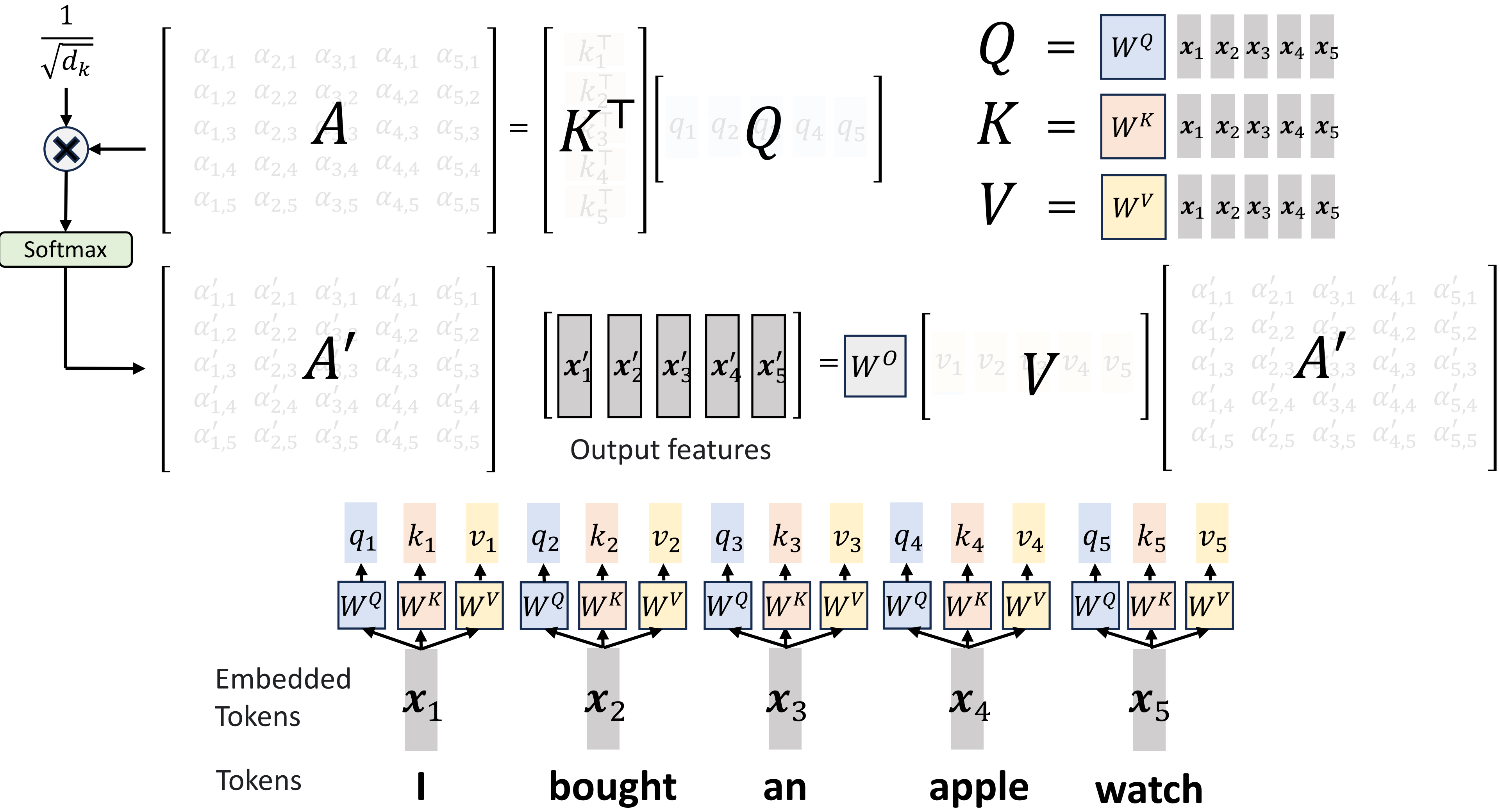

$\frac{1}{\sqrt{d_k}}$
Softmax
$\begin{bmatrix} \alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1} & \alpha_{5,1} \\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,2} & \alpha_{4,2} & \alpha_{5,2} \\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3} & \alpha_{5,3} \\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4} & \alpha_{5,4} \\ \alpha_{1,5} & \alpha_{2,5} & \alpha_{3,5} & \alpha_{4,5} & \alpha_{5,5} \end{bmatrix} = \begin{bmatrix} k_1^\top \\ k_2^\top \\ k_3^\top \\ k_4^\top \\ k_5^\top \end{bmatrix} \begin{bmatrix} q_1 & q_2 & q_3 & q_4 & q_5 \end{bmatrix}$
$\begin{bmatrix} \alpha'_{1,1} & \alpha'_{2,1} & \alpha'_{3,1} & \alpha'_{4,1} & \alpha'_{5,1} \\ \alpha'_{1,2} & \alpha'_{2,2} & \alpha'_{3,2} & \alpha'_{4,2} & \alpha'_{5,2} \\ \alpha'_{1,3} & \alpha'_{2,3} & \alpha'_{3,3} & \alpha'_{4,3} & \alpha'_{5,3} \\ \alpha'_{1,4} & \alpha'_{2,4} & \alpha'_{3,4} & \alpha'_{4,4} & \alpha'_{5,4} \\ \alpha'_{1,5} & \alpha'_{2,5} & \alpha'_{3,5} & \alpha'_{4,5} & \alpha'_{5,5} \end{bmatrix}$
q_1 k_1 v_1 q_2 k_2 v_2 q_3 k_3 v_3 q_4 k_4 v_4 q_5 k_5 v_5
W^Q W^K W^V
Embedded Tokens
$\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_3$ $\boldsymbol{x}_4$ $\boldsymbol{x}_5$
Tokens
I
bought
an
apple
watch


$\frac{1}{\sqrt{d_k}}$
Softmax
$k = [k^1, k^2, \cdots, k^{d_k}]^\top$
$q = [q^1, q^2, \cdots, q^{d_k}]^\top$
$E[k^i] = E[q^i] = 0$
$\mathrm{Var}[k^i] = \mathrm{Var}[q^i] = 1$
$k^\top q = \sum_{i=1}^{d_k} k^i q^i$
$\mathrm{Var}[k^\top q] = d_k$
Embedded Tokens
Tokens
I
bought
an
apple
watch



$\frac{1}{\sqrt{d_k}}$
Softmax
A
$=$
$K^\top$
Q
A'
q_1 k_1 v_1 q_2 k_2 v_2 q_3 k_3 v_3 q_4 k_4 v_4 q_5 k_5 v_5
W^Q W^K W^V
Embedded Tokens
$\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_3$ $\boldsymbol{x}_4$ $\boldsymbol{x}_5$
Tokens
I
bought
an
apple
watch


$\frac{1}{\sqrt{d_k}}$
Softmax
$A = K^\top Q$
A'
$[x'_1\ x'_2\ x'_3\ x'_4\ x'_5] = W^O [v_1\ v_2\ v_3\ v_4\ v_5]\, A'$
Output features
q_1 k_1 v_1 q_2 k_2 v_2 q_3 k_3 v_3 q_4 k_4 v_4 q_5 k_5 v_5
W^Q W^K W^V
Embedded Tokens
x_1 x_2 x_3 x_4 x_5
Tokens
I
bought
an
apple
watch



Softmax
Output features
Embedded Tokens
Tokens
I
bought
an
apple
watch

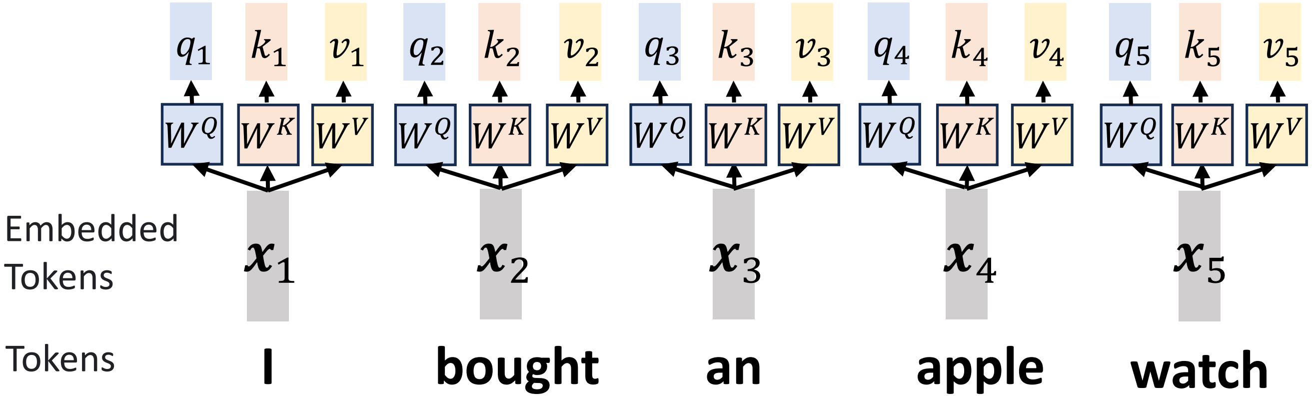
Single-head attention

$$\text{Attention}(Q, K, V) = V\,\text{softmax}\left(\frac{K^\top Q}{\sqrt{d_k}}\right)$$

$$Q = W^Q \; \boldsymbol{x}_1 \; \boldsymbol{x}_2 \; \boldsymbol{x}_3 \; \boldsymbol{x}_4 \; \boldsymbol{x}_5$$

$$K = W^K \; \boldsymbol{x}_1 \; \boldsymbol{x}_2 \; \boldsymbol{x}_3 \; \boldsymbol{x}_4 \; \boldsymbol{x}_5$$

$$V = W^V \; \boldsymbol{x}_1 \; \boldsymbol{x}_2 \; \boldsymbol{x}_3 \; \boldsymbol{x}_4 \; \boldsymbol{x}_5$$



Analogy for Q, K, V

- Library system
- Imagine you're looking for information on a specific topic (query)
- Each book in the library has a summary (key) that helps identify if it contains the information you're looking for
- Once you find a match between your query and a summary, you access the book to get the detailed information (value) you need
- Here, in Attention, we do a "soft match" across multiple values, e.g. get info from multiple books ("book 1 is most relevant, then book 2, then book 3, etc.")

$$\text{Attention}(Q, K, V) = V \text{ softmax}\left(\frac{K^\top Q}{\sqrt{d_k}}\right)$$

Single-head attention

$$\text{Attention}(Q, K, V) = V \,\text{softmax}\left(\frac{K^\top Q}{\sqrt{d_k}}\right)$$

$$Q = W^Q \; x_1 \; x_2 \; x_3 \; x_4 \; x_5$$

$$K = W^K \; x_1 \; x_2 \; x_3 \; x_4 \; x_5$$

$$V = W^V \; x_1 \; x_2 \; x_3 \; x_4 \; x_5$$

W_0^Q W_0^K W_0^V $\quad$ W_1^Q W_1^K W_1^V $\quad \cdots \quad$ W_{h-1}^Q W_{h-1}^K W_{h-1}^V

$W_i^Q \in R^{d_k \times d}$

$W_i^K \in R^{d_k \times d}$

$W_i^V \in R^{d_v \times d}$

Single-head attention

$$\text{Attention}(Q, K, V) = V\ \text{softmax}\left(\frac{K^\top Q}{\sqrt{d_k}}\right)$$

$$Q = W^Q\ x_1\ x_2\ x_3\ x_4\ x_5$$

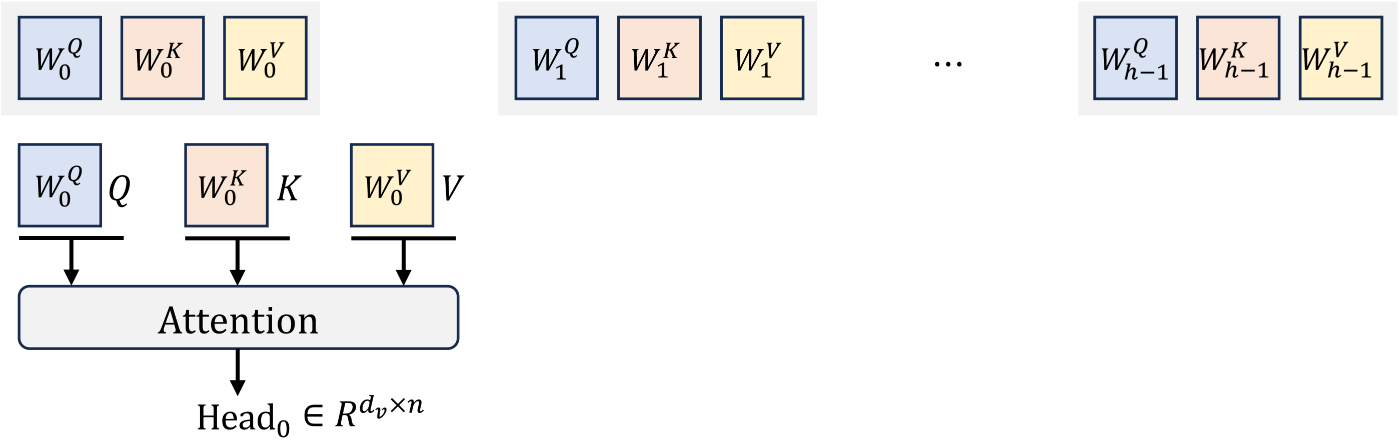
$$K = W^K\ x_1\ x_2\ x_3\ x_4\ x_5$$

$$V = W^V\ x_1\ x_2\ x_3\ x_4\ x_5$$

$W_i^Q \in R^{d_k \times d}$

$W_i^K \in R^{d_k \times d}$

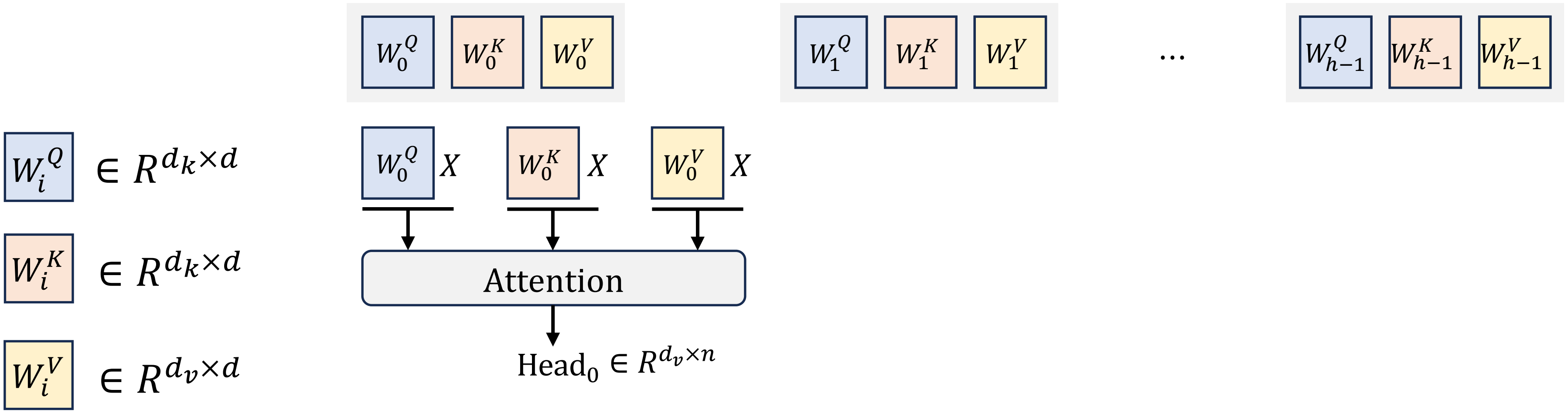
$W_i^V \in R^{d_v \times d}$



Single-head attention

$$\text{Attention}(Q, K, V) = V \,\text{softmax}\left(\frac{K^\top Q}{\sqrt{d_k}}\right)$$

$$X = \boldsymbol{x}_1\ \boldsymbol{x}_2\ \boldsymbol{x}_3\ \boldsymbol{x}_4\ \boldsymbol{x}_5$$



Single-head attention

$$\text{Attention}(Q, K, V) = V\ \text{softmax}\left(\frac{K^\top Q}{\sqrt{d_k}}\right)$$

$$Q = W^Q\ x_1\ x_2\ x_3\ x_4\ x_5$$

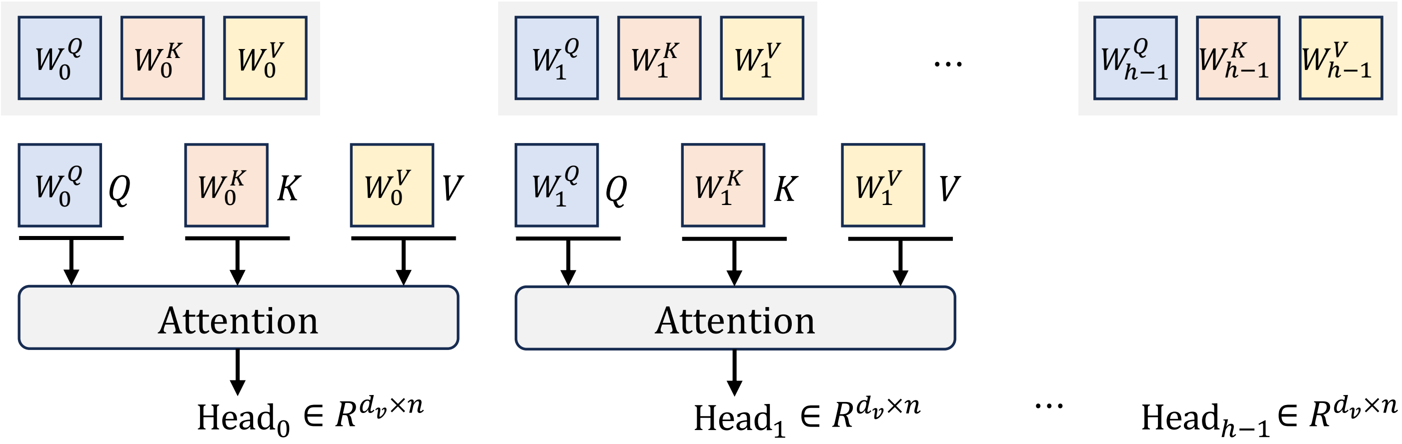
$$K = W^K\ x_1\ x_2\ x_3\ x_4\ x_5$$

$$V = W^V\ x_1\ x_2\ x_3\ x_4\ x_5$$

$W_i^Q \in R^{d_k \times d}$

$W_i^K \in R^{d_k \times d}$

$W_i^V \in R^{d_v \times d}$



Single-head attention

$$\text{Attention}(Q, K, V) = V \, \text{softmax}\left(\frac{K^\top Q}{\sqrt{d_k}}\right)$$

$$Q = W^Q \; x_1 \; x_2 \; x_3 \; x_4 \; x_5$$

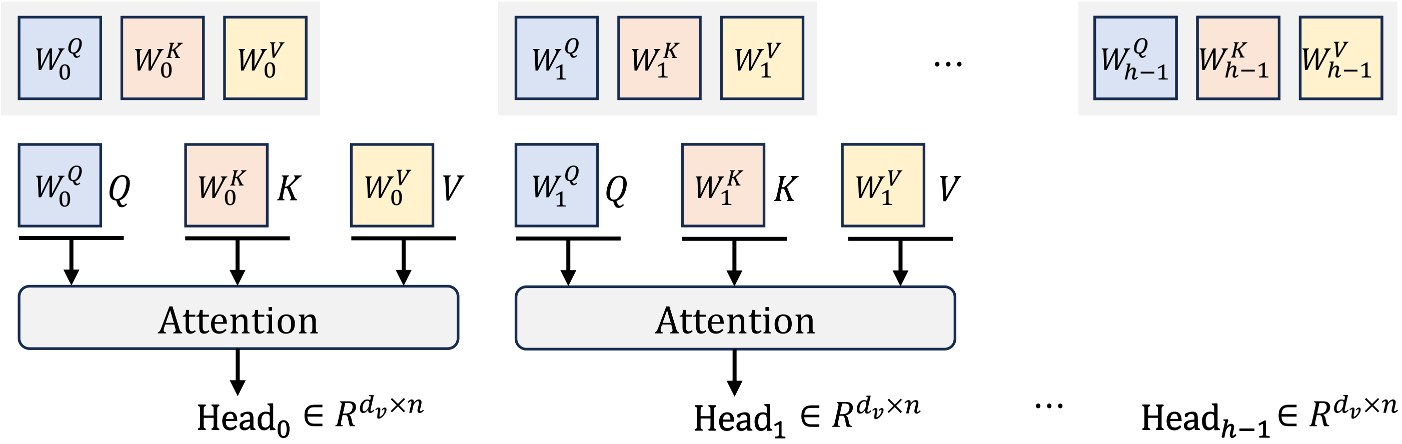
$$K = W^K \; x_1 \; x_2 \; x_3 \; x_4 \; x_5$$

$$V = W^V \; x_1 \; x_2 \; x_3 \; x_4 \; x_5$$

$W_i^Q \in R^{d_k \times d}$

$W_i^K \in R^{d_k \times d}$

$W_i^V \in R^{d_v \times d}$



Single-head attention

$$\mathrm{Attention}(Q,K,V) = V\ \mathrm{softmax}\left(\frac{K^\top Q}{\sqrt{d_k}}\right)$$

$$Q = W^Q\ \boldsymbol{x}_1\ \boldsymbol{x}_2\ \boldsymbol{x}_3\ \boldsymbol{x}_4\ \boldsymbol{x}_5$$
$$K = W^K\ \boldsymbol{x}_1\ \boldsymbol{x}_2\ \boldsymbol{x}_3\ \boldsymbol{x}_4\ \boldsymbol{x}_5$$
$$V = W^V\ \boldsymbol{x}_1\ \boldsymbol{x}_2\ \boldsymbol{x}_3\ \boldsymbol{x}_4\ \boldsymbol{x}_5$$

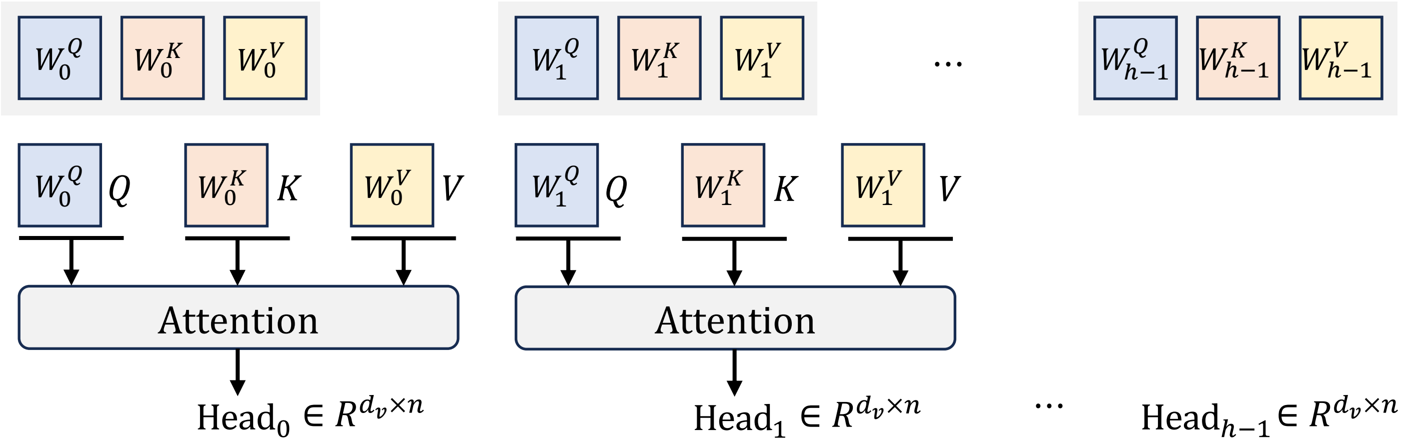
Multi-head attention

$W^O \in R^{d \times h d_v}$

$W_i^Q \in R^{d_k \times d}$

$W_i^K \in R^{d_k \times d}$

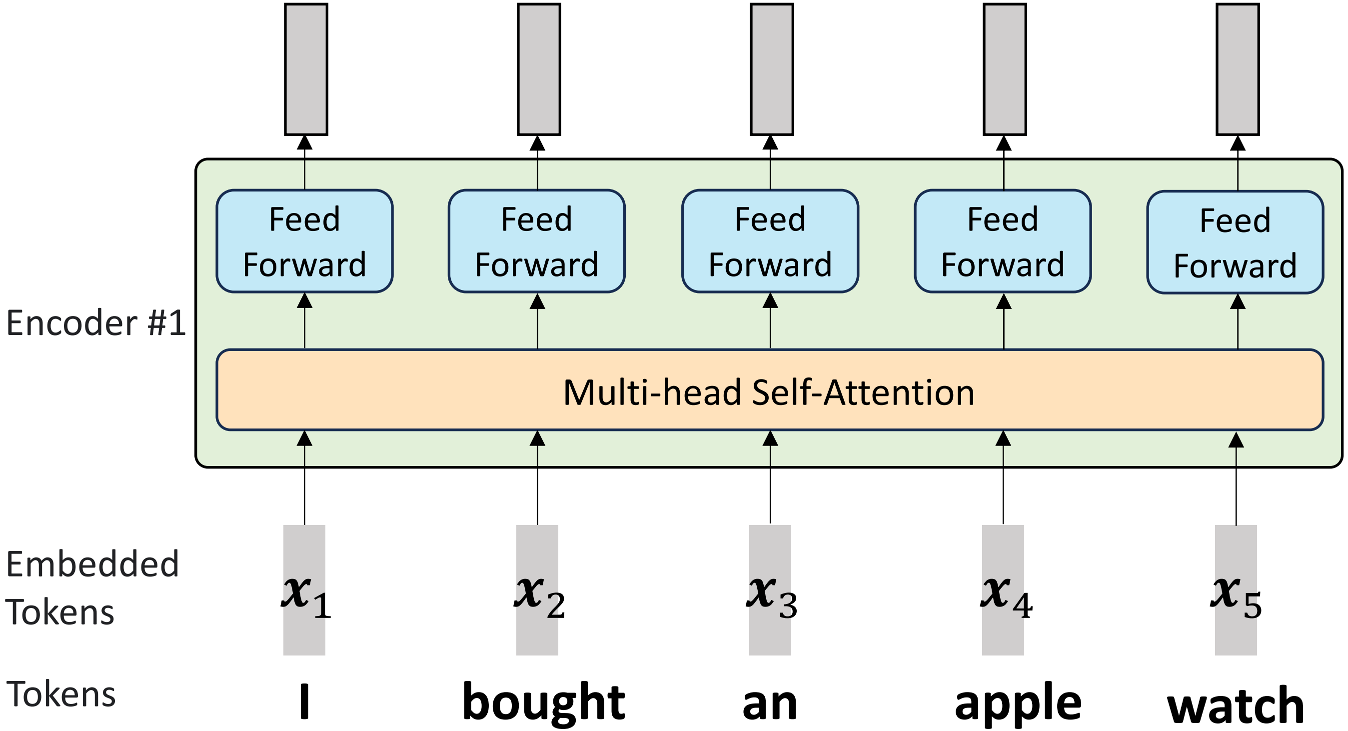
$W_i^V \in R^{d_v \times d}$



$$\mathrm{MultiHeadedAttention}(Q,K,V) = W^O \begin{bmatrix} \mathrm{Head}_0 \\ \mathrm{Head}_1 \\ \vdots \\ \mathrm{Head}_{h-1} \end{bmatrix}$$

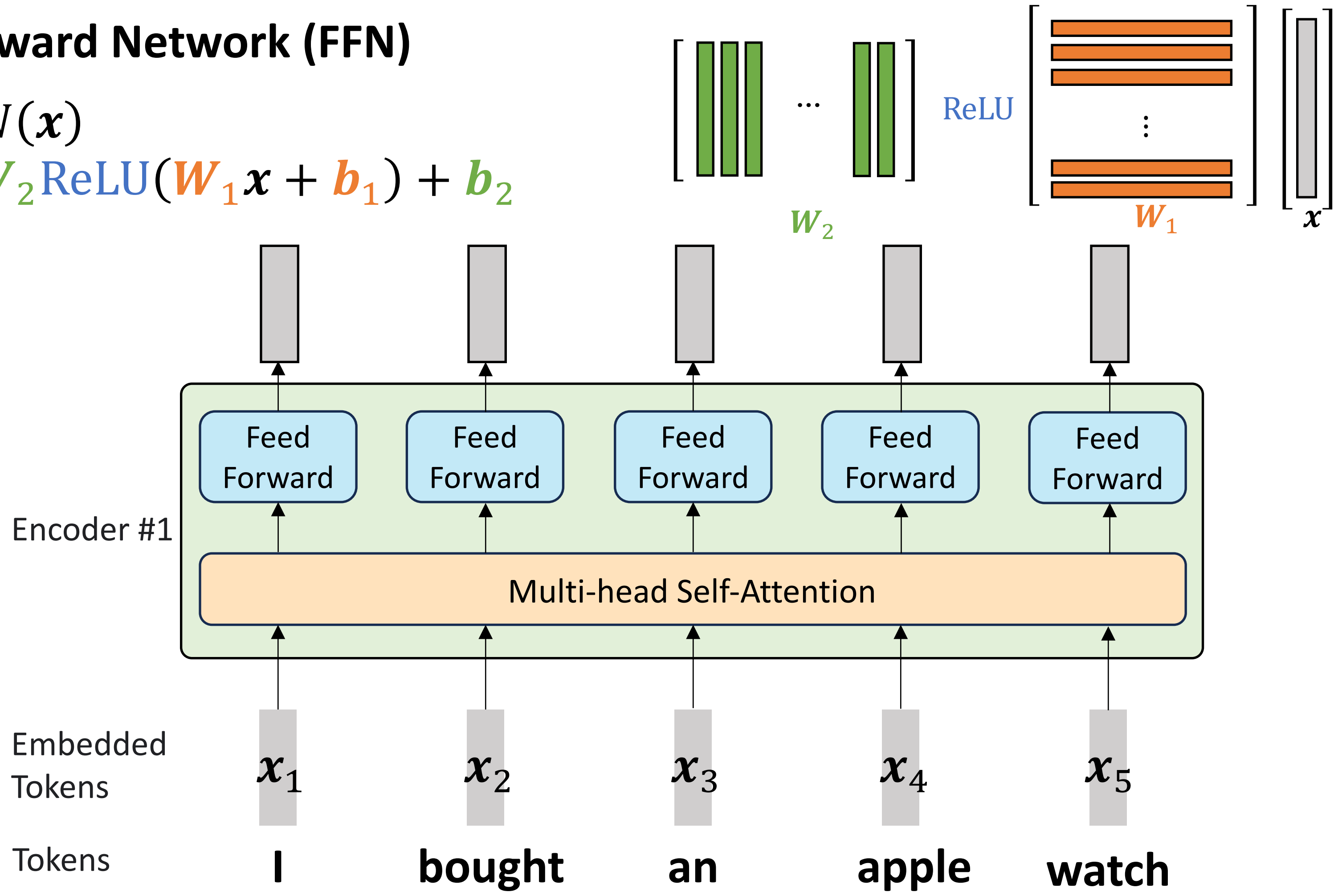
Feed Forward Network (FFN)

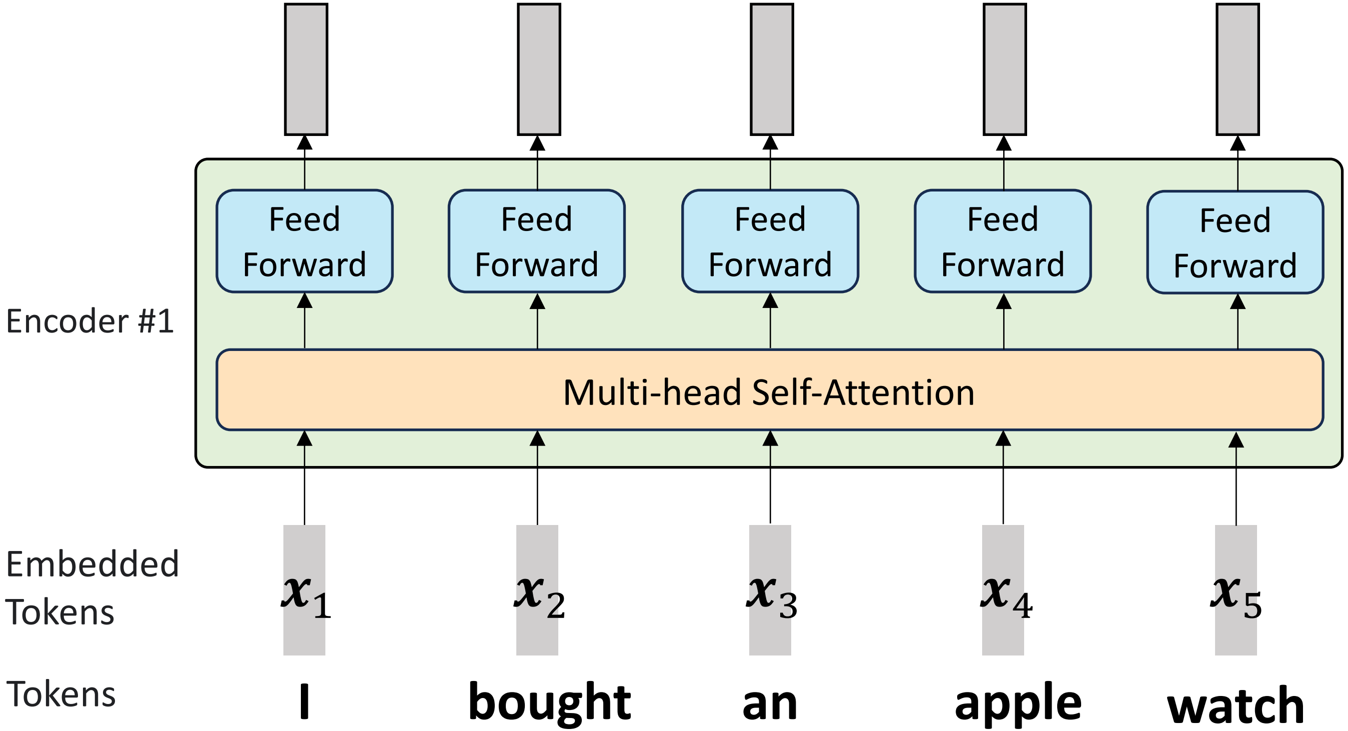
$$FFN(\boldsymbol{x}) = \boldsymbol{W}_2 \mathrm{ReLU}(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2$$



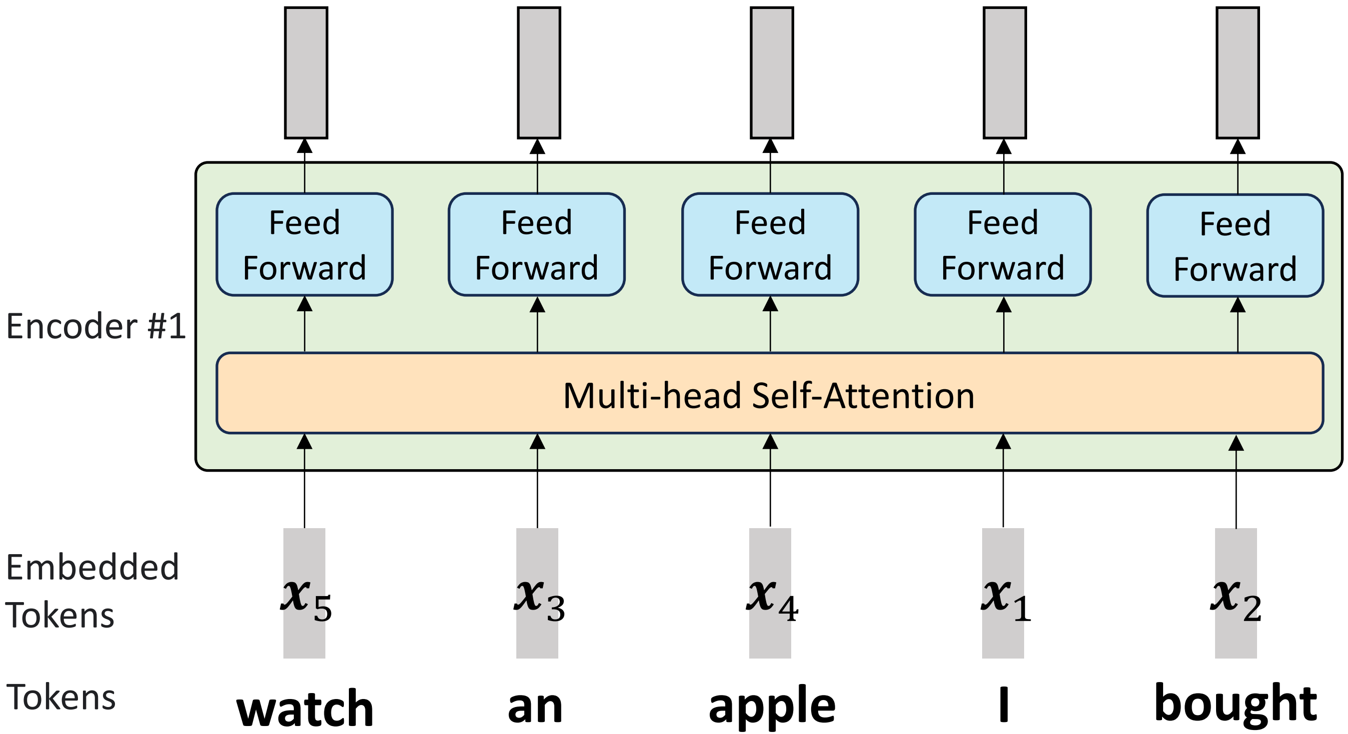
Feed Forward Network (FFN)

$$FFN(\boldsymbol{x}) = \boldsymbol{W}_2 \mathrm{ReLU}(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2$$

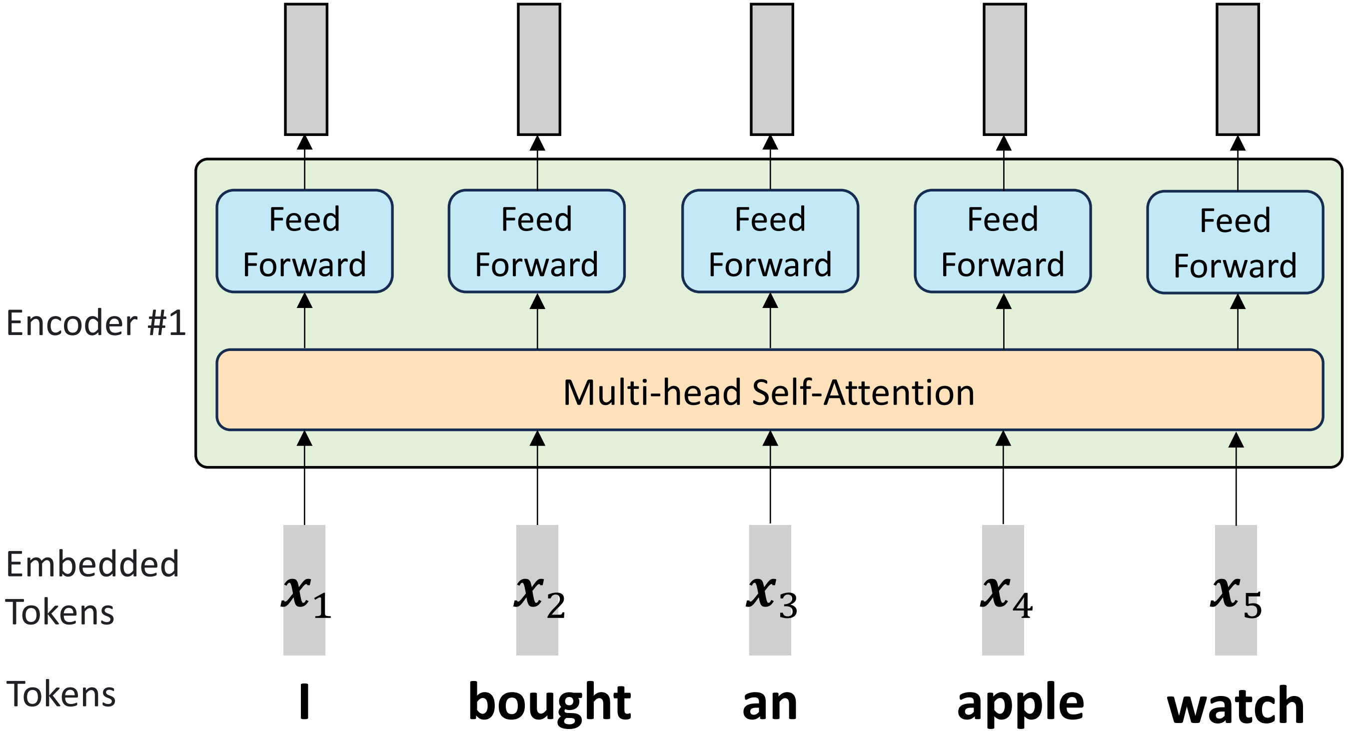




Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Encoder #1
Multi-head Self-Attention
Embedded Tokens
x_1
x_2
x_3
x_4
x_5
Tokens
I
bought
an
apple
watch

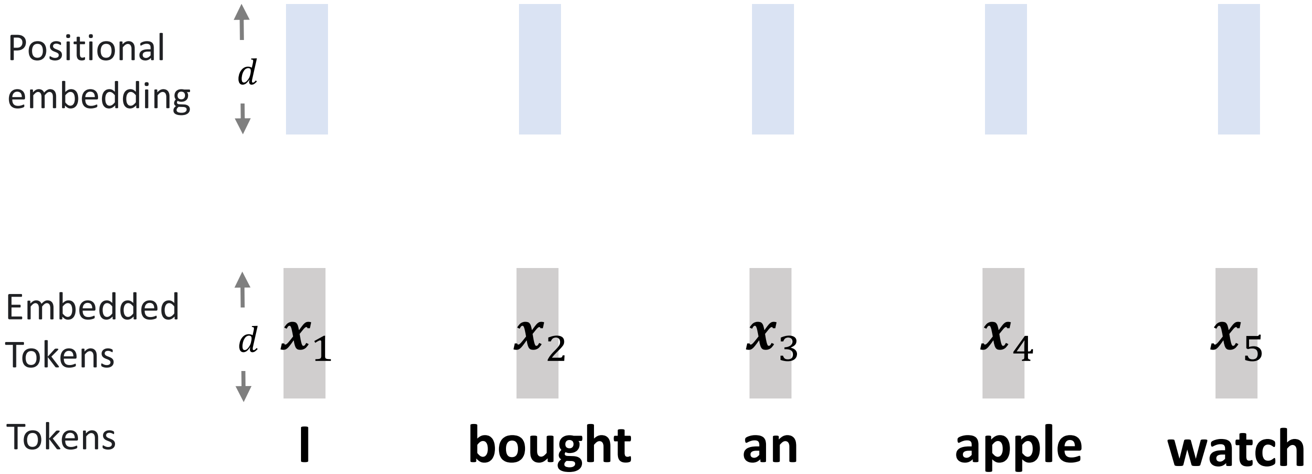

Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Encoder #1
Multi-head Self-Attention
Embedded Tokens
x_5
x_3
x_4
x_1
x_2
Tokens
watch
an
apple
I
bought

Positional encoding



Positional encoding

Position k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Dimension 2^3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	← Slow oscillating
2^2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	
2^1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	
2^0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	← Fast oscillating



Positional encoding

Position k

Angular frequency

$w_i = N^{-2i/d}$

$N = 100{,}000$

$$d \left[\begin{array}{c} \sin(w_0 k) \\ \cos(w_0 k) \\ \sin(w_1 k) \\ \cos(w_1 k) \\ \vdots \\ \vdots \\ \sin\left(w_{\frac{d}{2}-1} k\right) \\ \cos\left(w_{\frac{d}{2}-1} k\right) \end{array} \right]$$

← Fast oscillating

← Slow oscillating

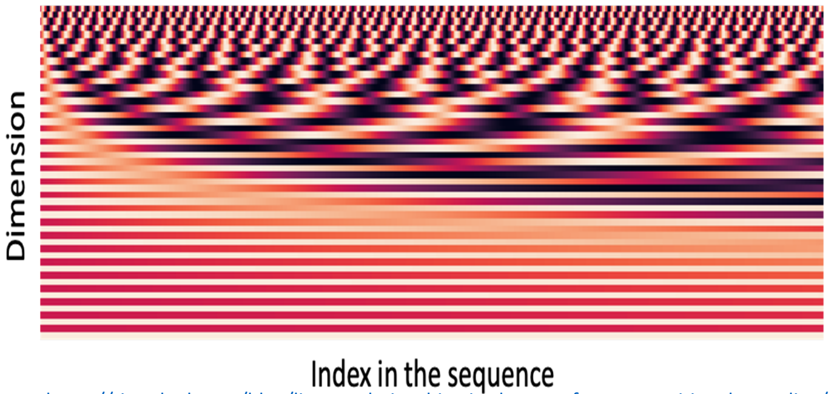


Image: https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

Positional embedding d

Embedded Tokens d $\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_3$ $\boldsymbol{x}_4$ $\boldsymbol{x}_5$

Tokens **I** **bought** **an** **apple** **watch**

Positional encoding

- Normalized Range
- Unique identifier, unlimited length
- Relative positions as linear transform

Position k

Angular frequency

$$w_i = N^{-2i/d}$$

$$N = 100{,}000$$

$$d \left\{ \begin{bmatrix} \sin(w_0 k) \\ \cos(w_0 k) \\ \sin(w_1 k) \\ \cos(w_1 k) \\ \vdots \\ \vdots \\ \sin\left(w_{\frac{d}{2}-1} k\right) \\ \cos\left(w_{\frac{d}{2}-1} k\right) \end{bmatrix} \right.$$

$$\begin{bmatrix} \sin(w_i(k + \Delta k)) \\ \cos(w_i(k + \Delta k)) \end{bmatrix} = \begin{bmatrix} \sin(w_i k)\cos(w_i \Delta k) + \cos(w_i k)\sin(w_i \Delta k) \\ \cos(w_i k)\cos(w_i \Delta k) - \sin(w_i k)\sin(w_i \Delta k) \end{bmatrix}$$

Positional embedding	d					
Embedded Tokens	d	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$\boldsymbol{x}_5$
Tokens		**I**	**bought**	**an**	**apple**	**watch**

Positional encoding

- Normalized Range
- Unique identifier, unlimited length
- Relative positions as linear transform

Position k

Angular frequency

$$w_i = N^{-2i/d}$$

$$N = 100{,}000$$

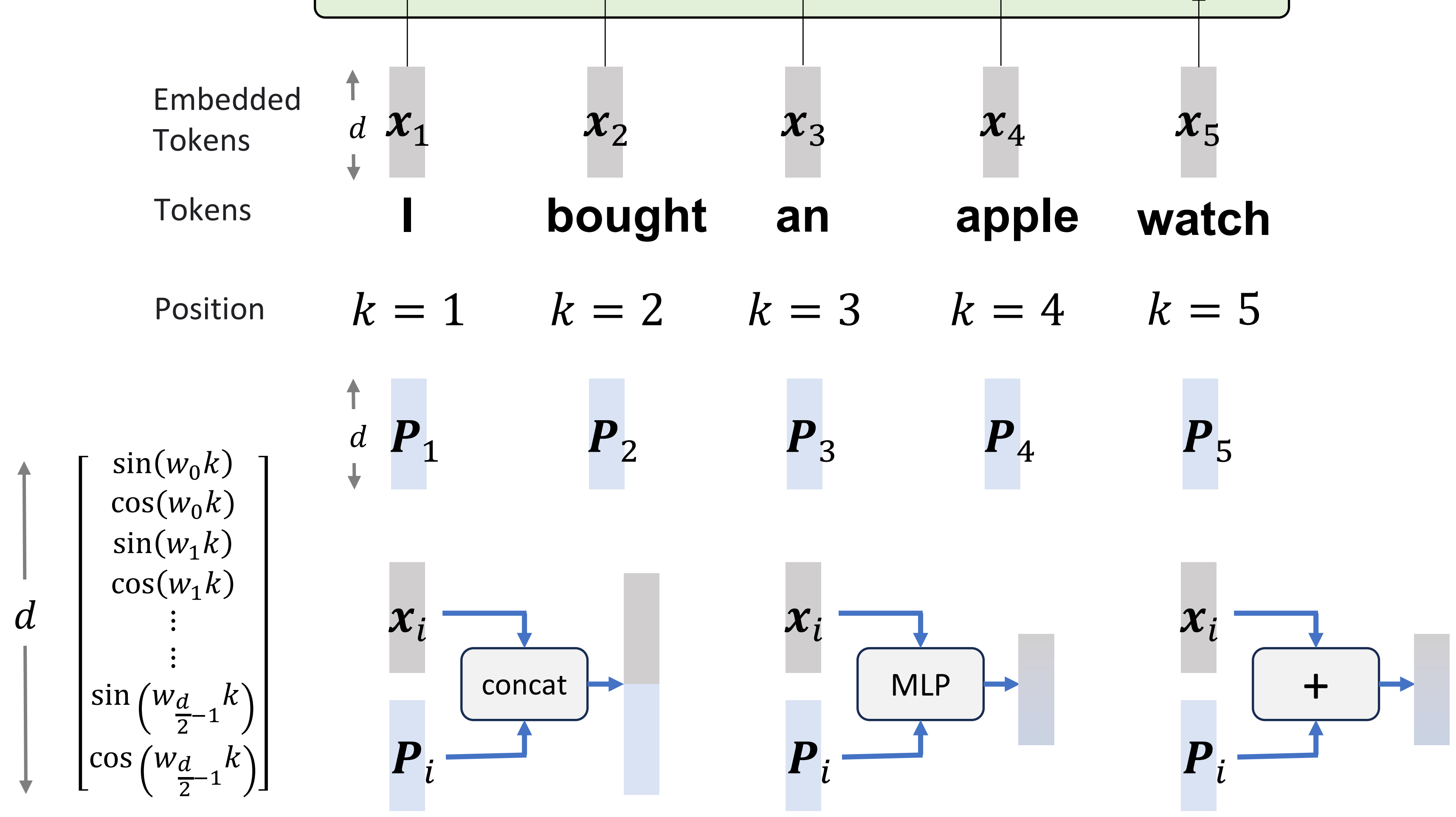
$$d \left\{ \begin{bmatrix} \sin(w_0 k) \\ \cos(w_0 k) \\ \sin(w_1 k) \\ \cos(w_1 k) \\ \vdots \\ \vdots \\ \sin\left(w_{\frac{d}{2}-1} k\right) \\ \cos\left(w_{\frac{d}{2}-1} k\right) \end{bmatrix} \right.$$

$$\begin{bmatrix} \sin(w_i(k + \Delta k)) \\ \cos(w_i(k + \Delta k)) \end{bmatrix} = \begin{bmatrix} \sin(w_i k)\cos(w_i \Delta k) + \cos(w_i k)\sin(w_i \Delta k) \\ \cos(w_i k)\cos(w_i \Delta k) - \sin(w_i k)\sin(w_i \Delta k) \end{bmatrix}$$

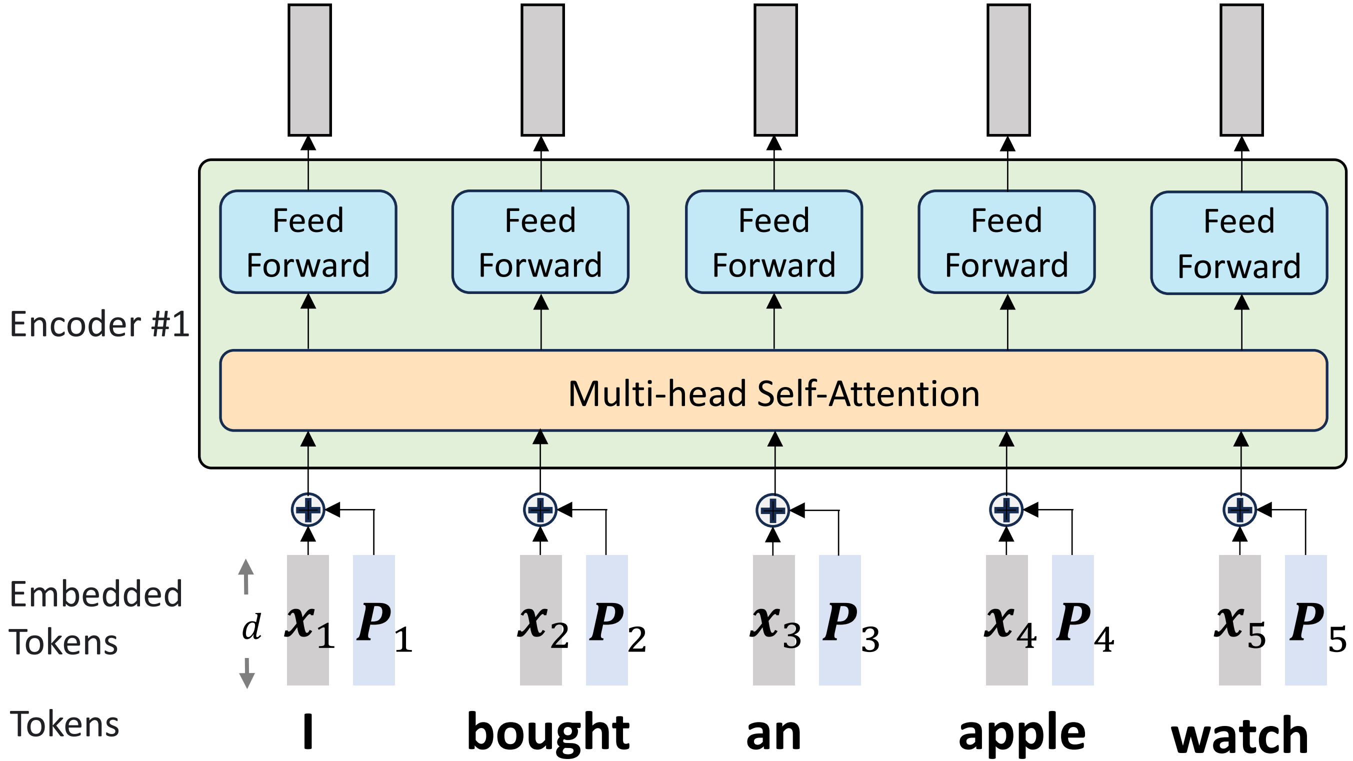
$$= \begin{bmatrix} \cos(w_i \Delta k) & \sin(w_i \Delta k) \\ -\sin(w_i \Delta k) & \cos(w_i \Delta k) \end{bmatrix} \begin{bmatrix} \sin(w_i k) \\ \cos(w_i k) \end{bmatrix}$$

$$P_{k+\Delta k} = M P_k$$

Positional embedding (d)	$\boldsymbol{P}_1$	$\boldsymbol{P}_2$	$\boldsymbol{P}_3$	$\boldsymbol{P}_4$	$\boldsymbol{P}_5$
Embedded Tokens (d)	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$\boldsymbol{x}_5$
Tokens	**I**	**bought**	**an**	**apple**	**watch**



Embedded Tokens
d
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$
$\boldsymbol{x}_4$
$\boldsymbol{x}_5$
Tokens
I
bought
an
apple
watch
Position
$k = 1$
$k = 2$
$k = 3$
$k = 4$
$k = 5$
d
$\boldsymbol{P}_1$
$\boldsymbol{P}_2$
$\boldsymbol{P}_3$
$\boldsymbol{P}_4$
$\boldsymbol{P}_5$
d
$\begin{bmatrix} \sin(w_0 k) \\ \cos(w_0 k) \\ \sin(w_1 k) \\ \cos(w_1 k) \\ \vdots \\ \vdots \\ \sin\left(w_{\frac{d}{2}-1} k\right) \\ \cos\left(w_{\frac{d}{2}-1} k\right) \end{bmatrix}$
$\boldsymbol{x}_i$
$\boldsymbol{P}_i$
concat
$\boldsymbol{x}_i$
$\boldsymbol{P}_i$
MLP
$\boldsymbol{x}_i$
$\boldsymbol{P}_i$
+


Encoder #1
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Multi-head Self-Attention
Embedded Tokens
d
x_1 P_1
x_2 P_2
x_3 P_3
x_4 P_4
x_5 P_5
Tokens
I
bought
an
apple
watch

Positional encoding

Position k

Angular frequency

$w_i = N^{-2i/d}$

$N = 100{,}000$

$$d \begin{bmatrix} \sin(w_0 k) \\ \cos(w_0 k) \\ \sin(w_1 k) \\ \cos(w_1 k) \\ \vdots \\ \vdots \\ \sin\left(w_{\frac{d}{2}-1} k\right) \\ \cos\left(w_{\frac{d}{2}-1} k\right) \end{bmatrix}$$

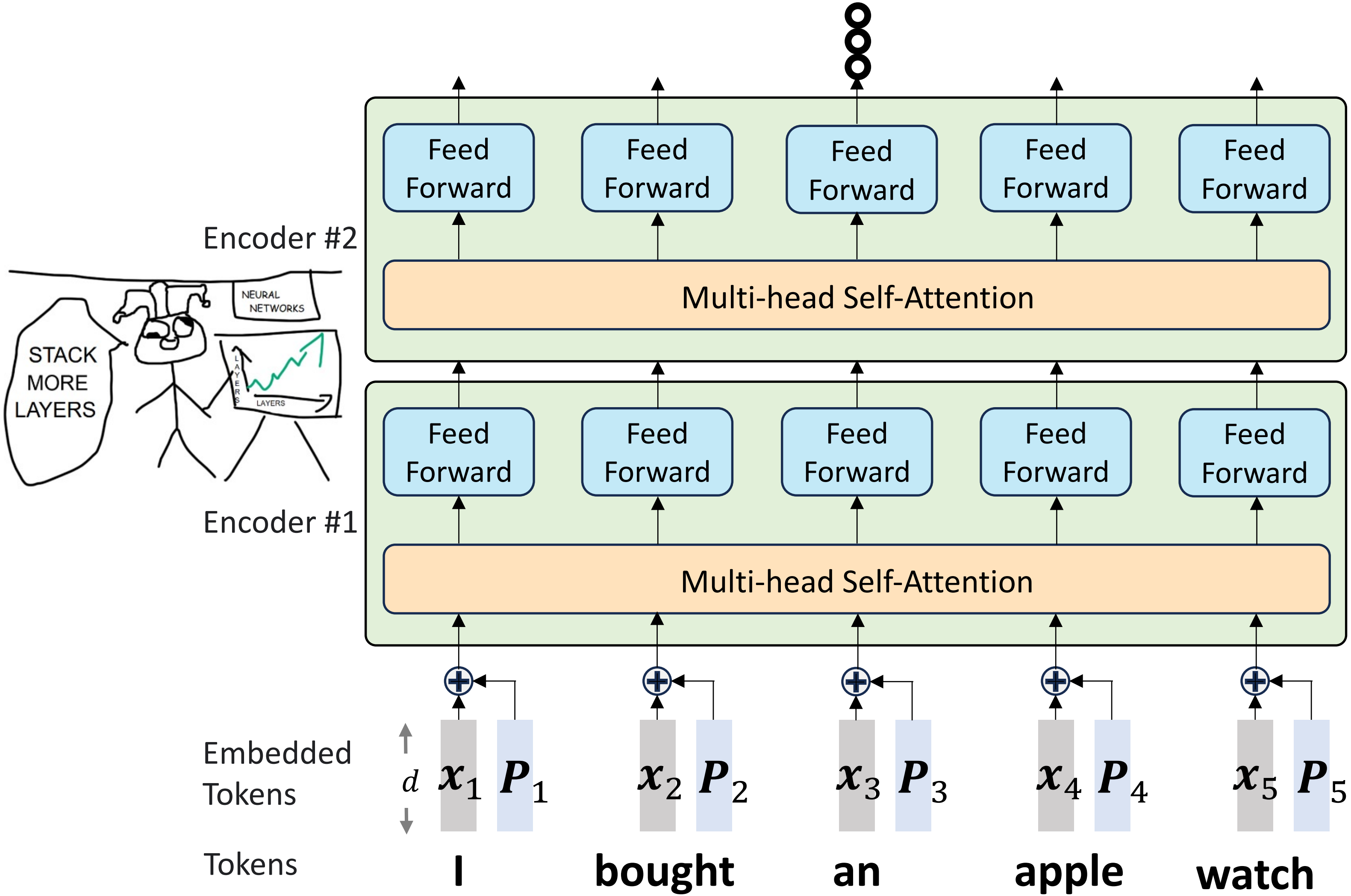
Sinusoidal positional encoding

Relative positional encoding

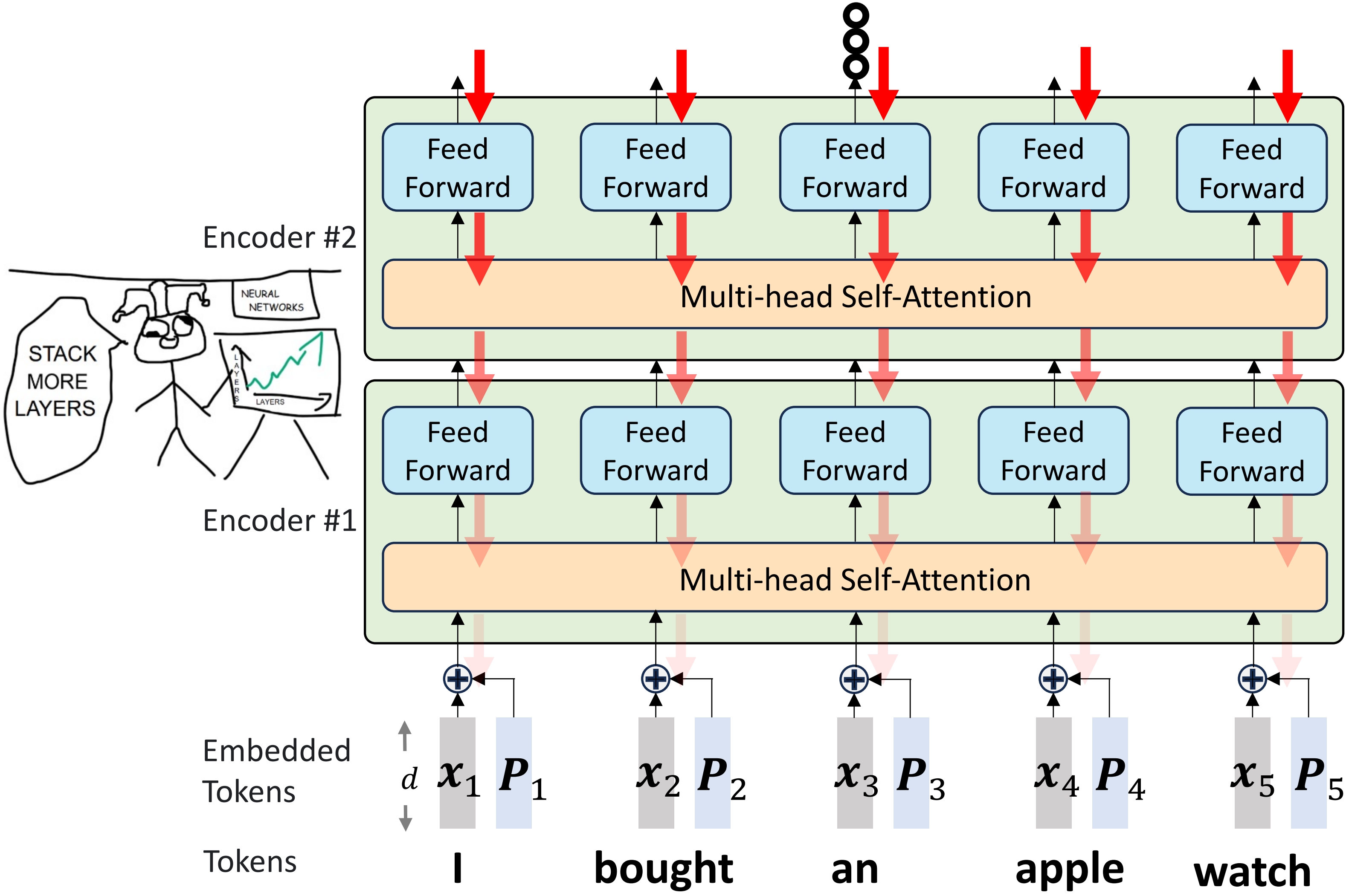
KERPLE RoPE CoPE

NoPE YaRN FIRE

Positional embedding	d $\boldsymbol{P}_1$	$\boldsymbol{P}_2$	$\boldsymbol{P}_3$	$\boldsymbol{P}_4$	$\boldsymbol{P}_5$
Embedded Tokens	d $\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$\boldsymbol{x}_5$
Tokens	**I**	**bought**	**an**	**apple**	**watch**

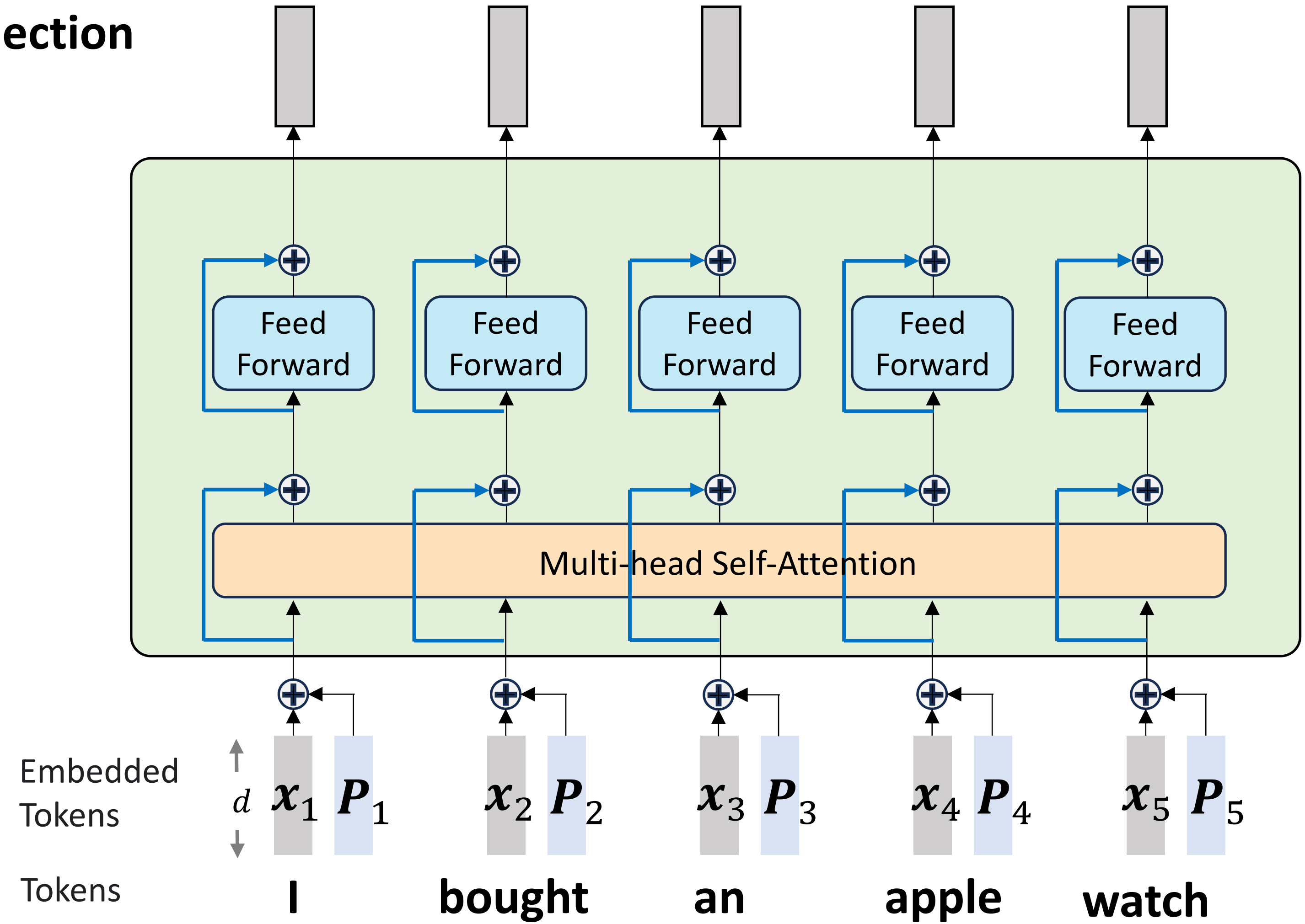


Encoder #2
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Multi-head Self-Attention
STACK MORE LAYERS
NEURAL NETWORKS
LAYERS
LAYERS
Encoder #1
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Multi-head Self-Attention
Embedded Tokens
d
x_1 P_1
x_2 P_2
x_3 P_3
x_4 P_4
x_5 P_5
Tokens
I
bought
an
apple
watch

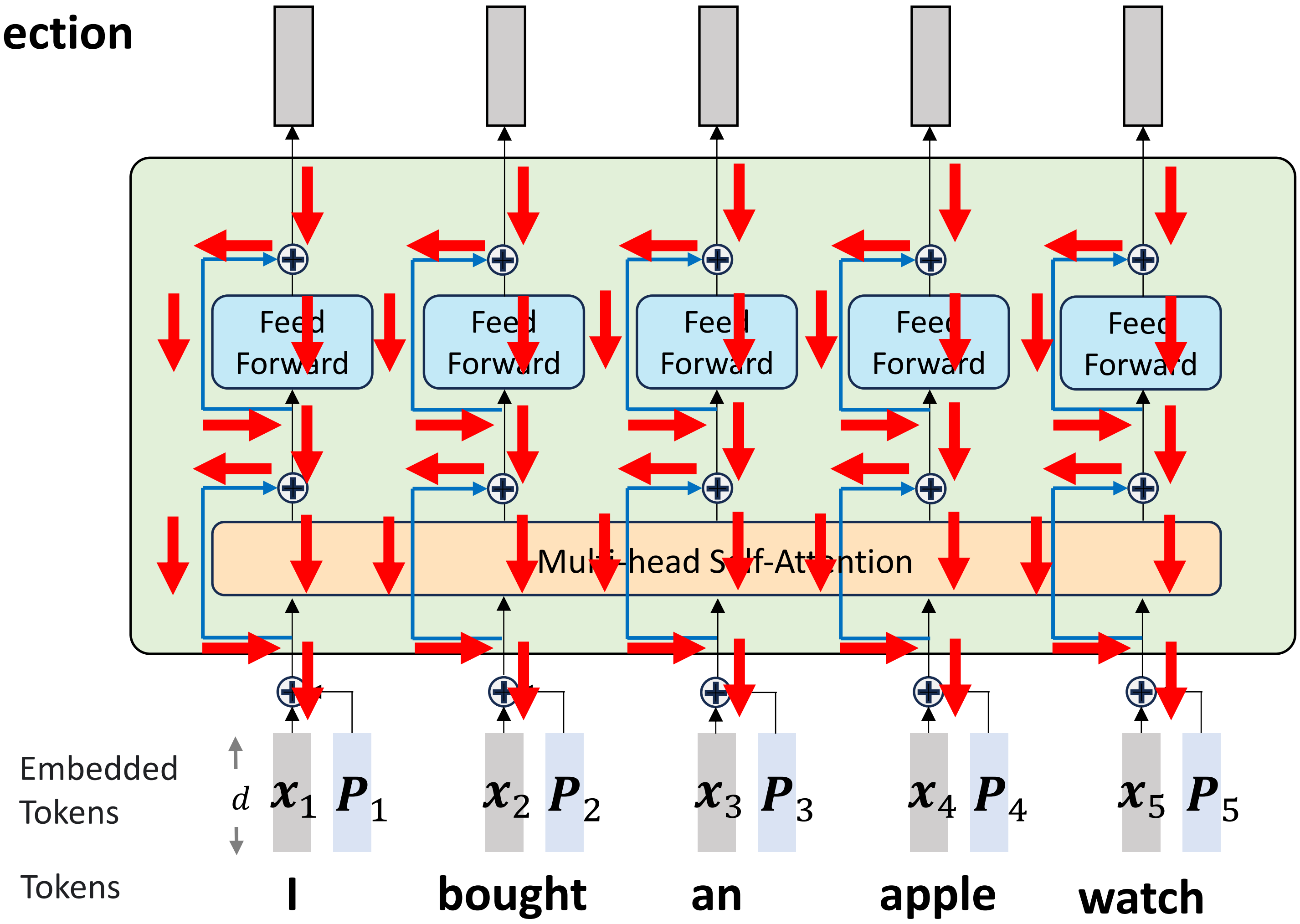


Encoder #2
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Multi-head Self-Attention
Encoder #1
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Feed Forward
Multi-head Self-Attention
NEURAL NETWORKS
STACK MORE LAYERS
LAYERS
LAYERS
Embedded Tokens
d
x_1 P_1
x_2 P_2
x_3 P_3
x_4 P_4
x_5 P_5
Tokens
I
bought
an
apple
watch

Residual connection

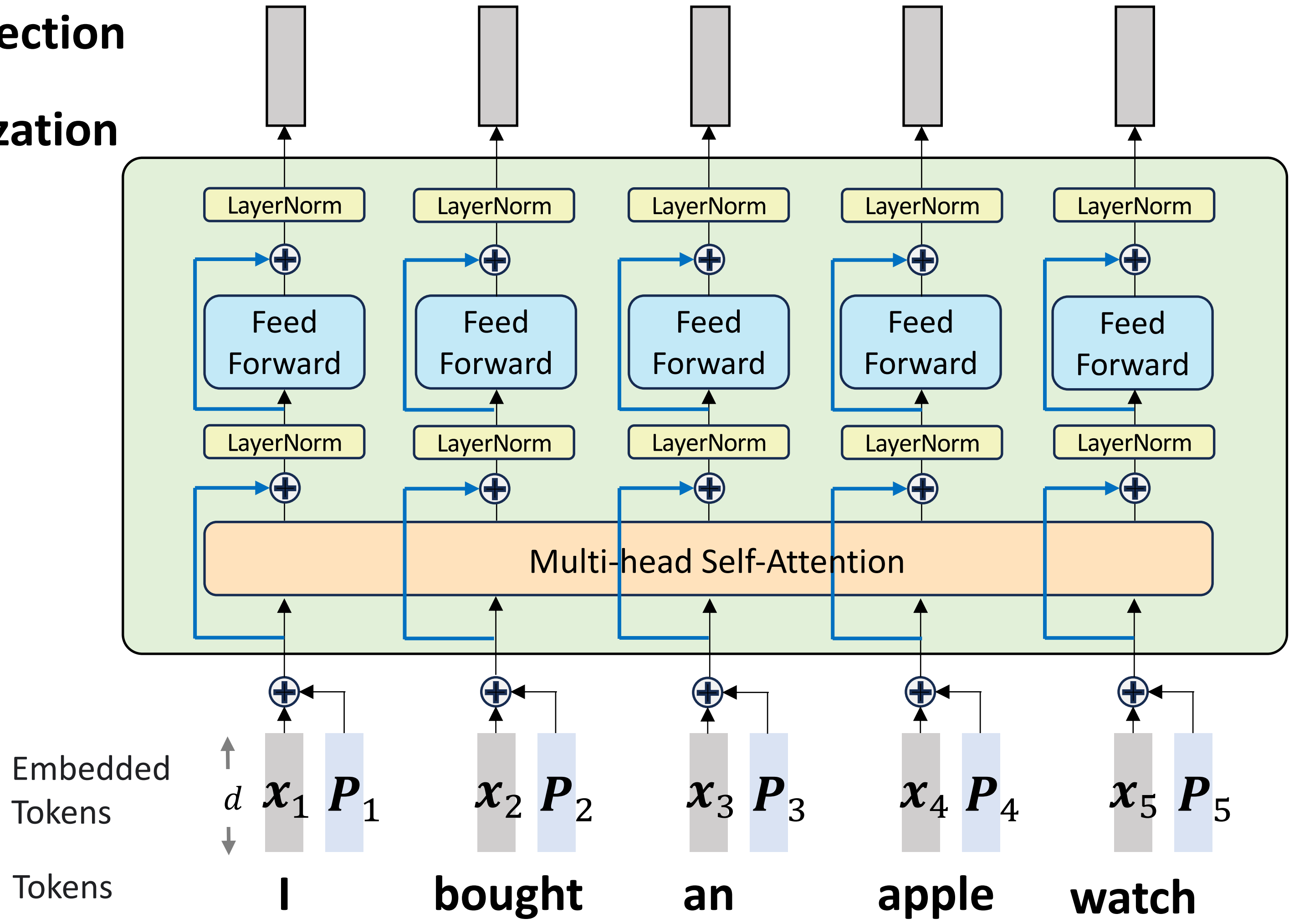


Residual connection



Residual connection

Layer normalization



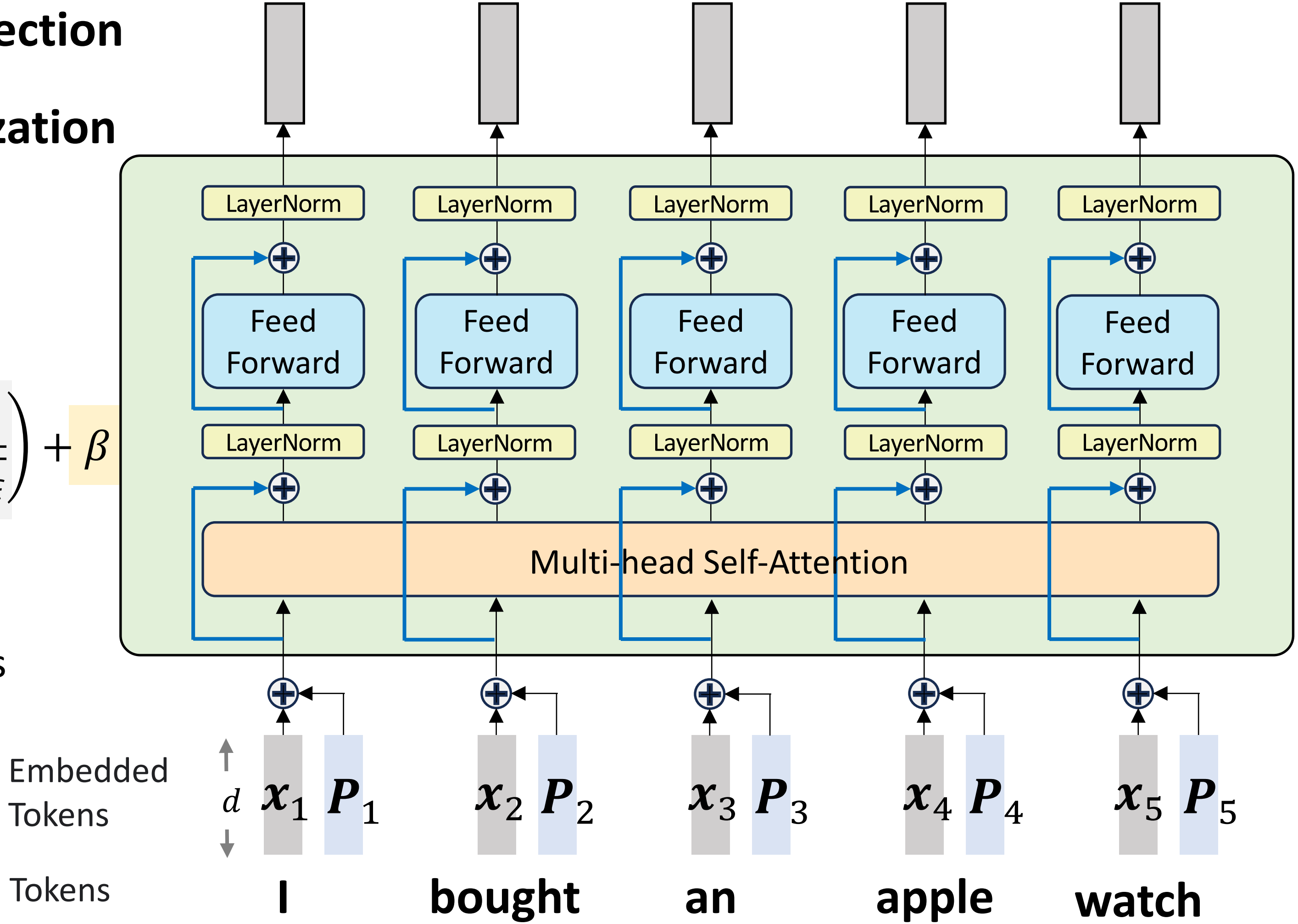
Residual connection

Layer normalization

$$\mathrm{LayerNorm}(\boldsymbol{x}) = \gamma\left(\frac{\boldsymbol{x} - \mathrm{mean}(\boldsymbol{x})}{\sqrt{\mathrm{Variance}(\mathbf{x}) + \epsilon}}\right) + \beta$$

$\gamma, \beta \in R$

Learnable parameters



Residual connection

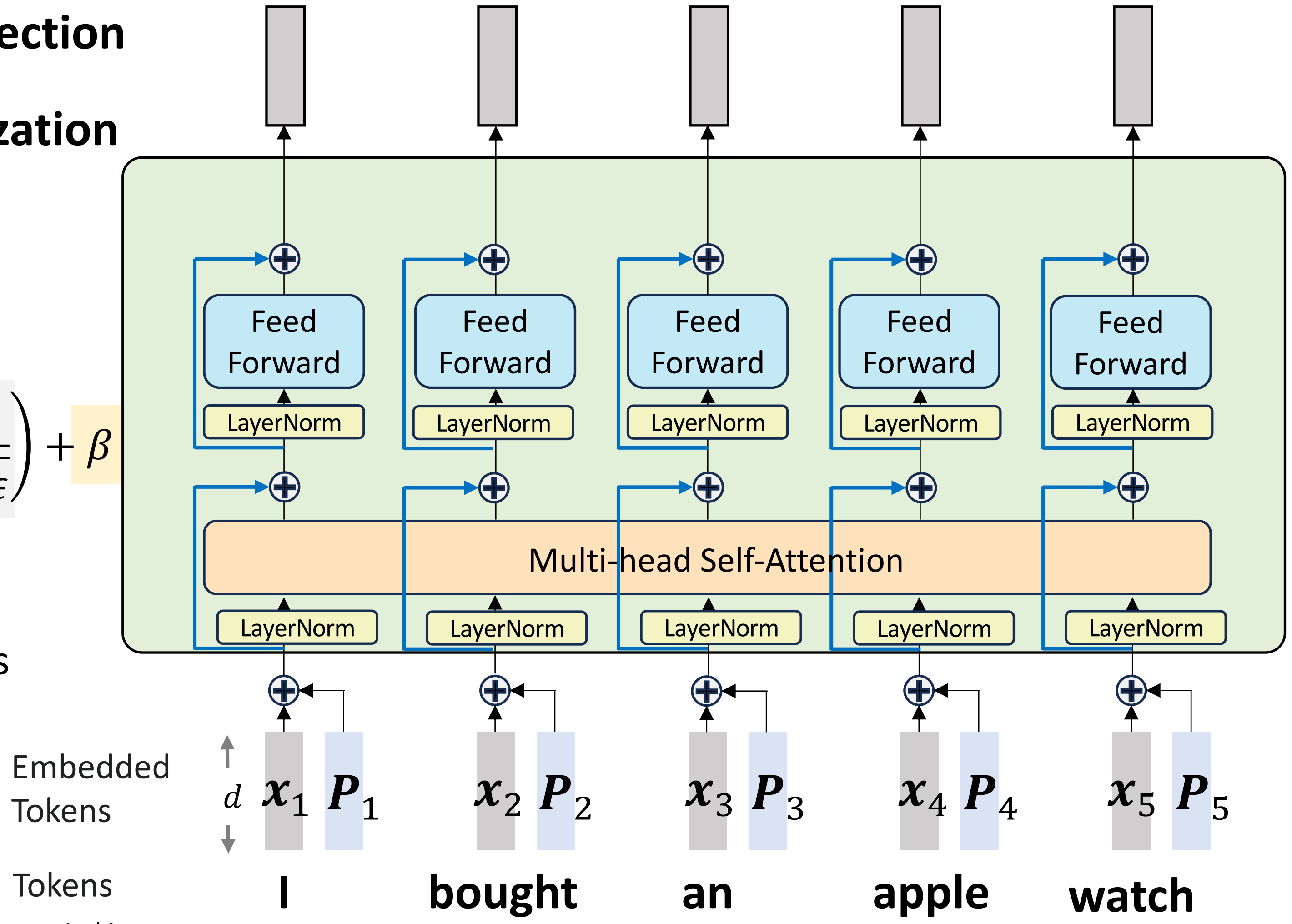
Layer normalization

$$\mathrm{LayerNorm}(\boldsymbol{x}) =$$

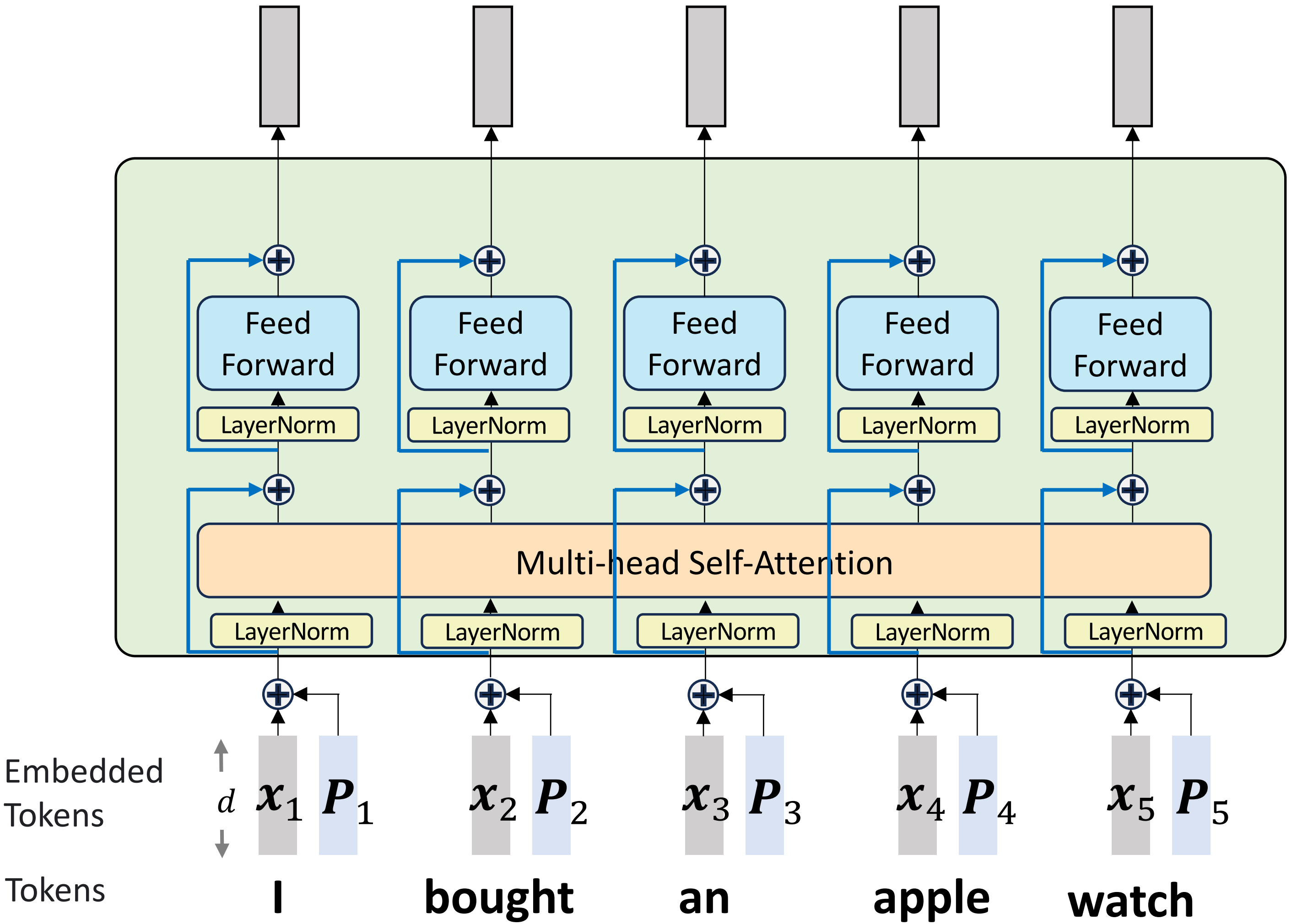
$$\gamma\left(\frac{\boldsymbol{x} - \mathrm{mean}(\boldsymbol{x})}{\sqrt{\mathrm{Variance}(\mathbf{x}) + \epsilon}}\right) + \beta$$

$\gamma, \beta \in R$

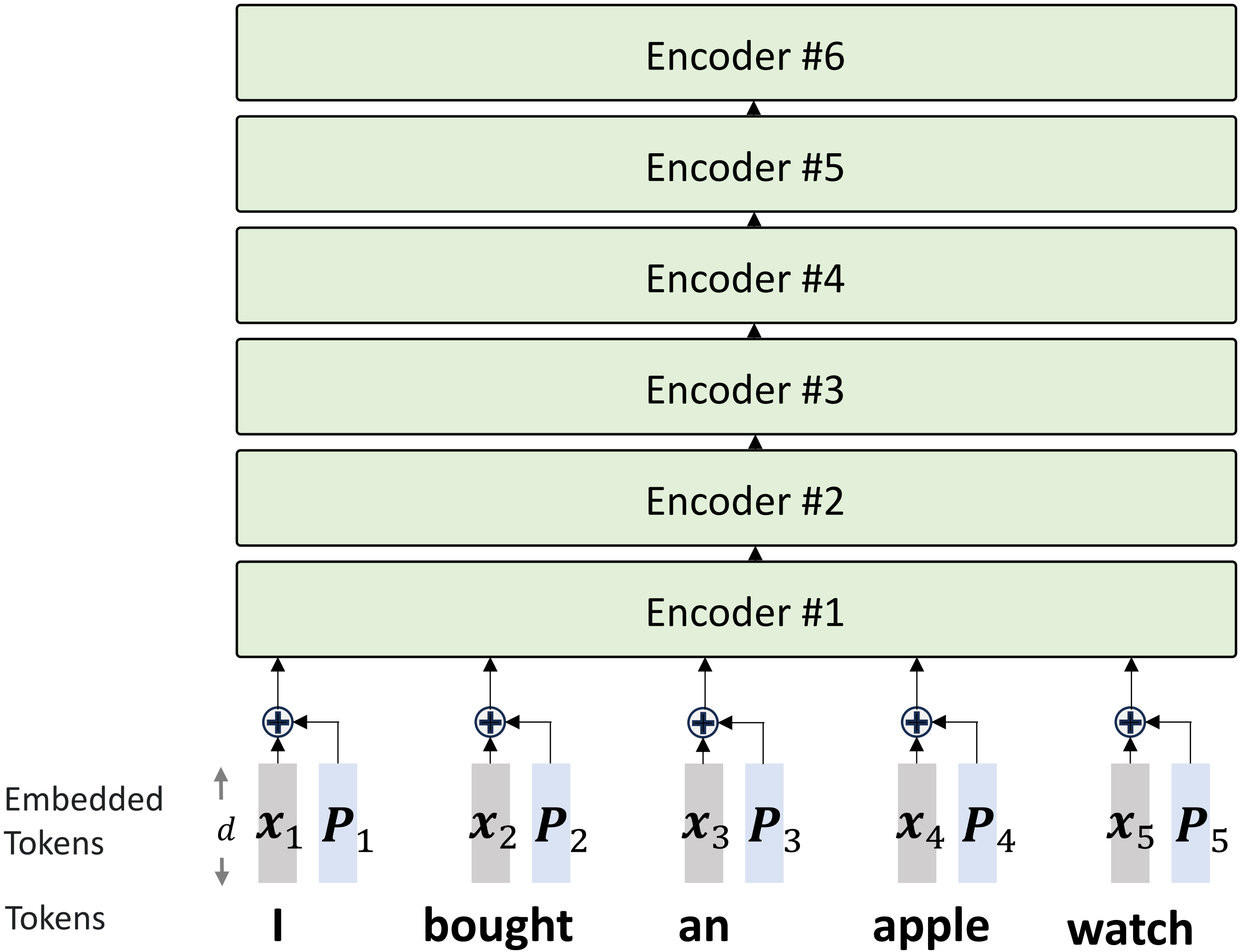
Learnable parameters



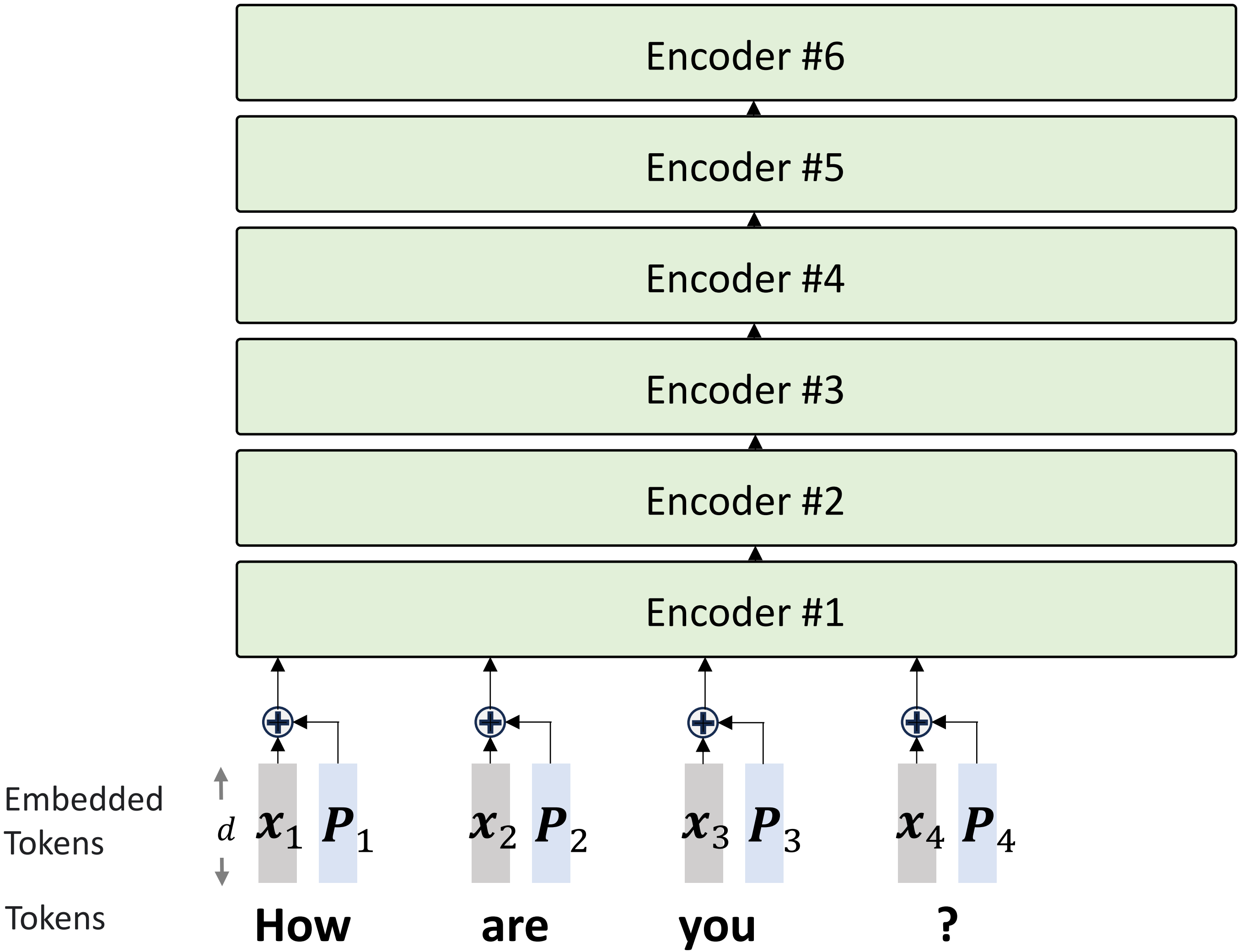
[Xiong et al. 2020]
On Layer Normalization in the Transformer Architecture



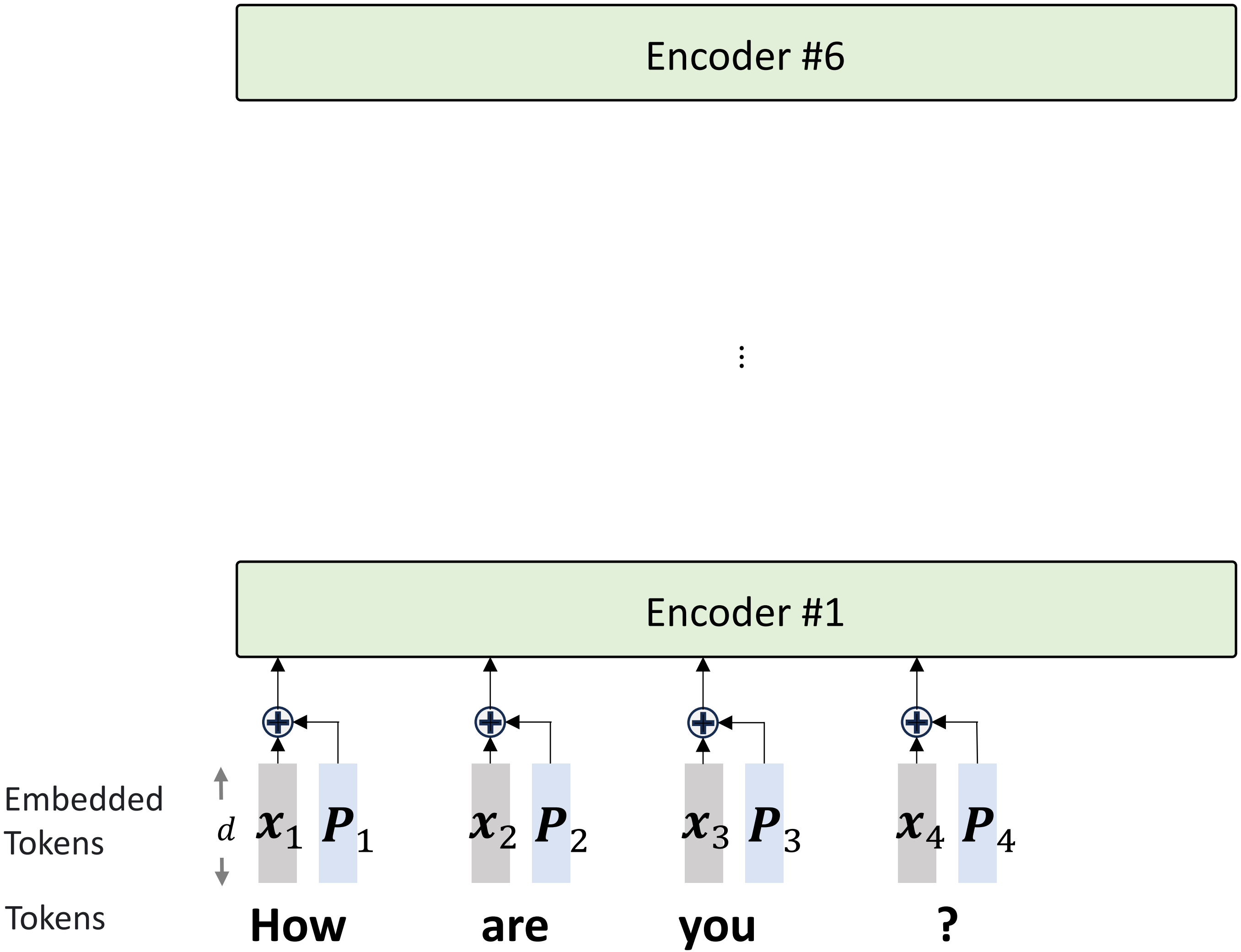
Feed Forward
LayerNorm
Multi-head Self-Attention
LayerNorm
Embedded Tokens
d
x_1 P_1
x_2 P_2
x_3 P_3
x_4 P_4
x_5 P_5
Tokens
I
bought
an
apple
watch



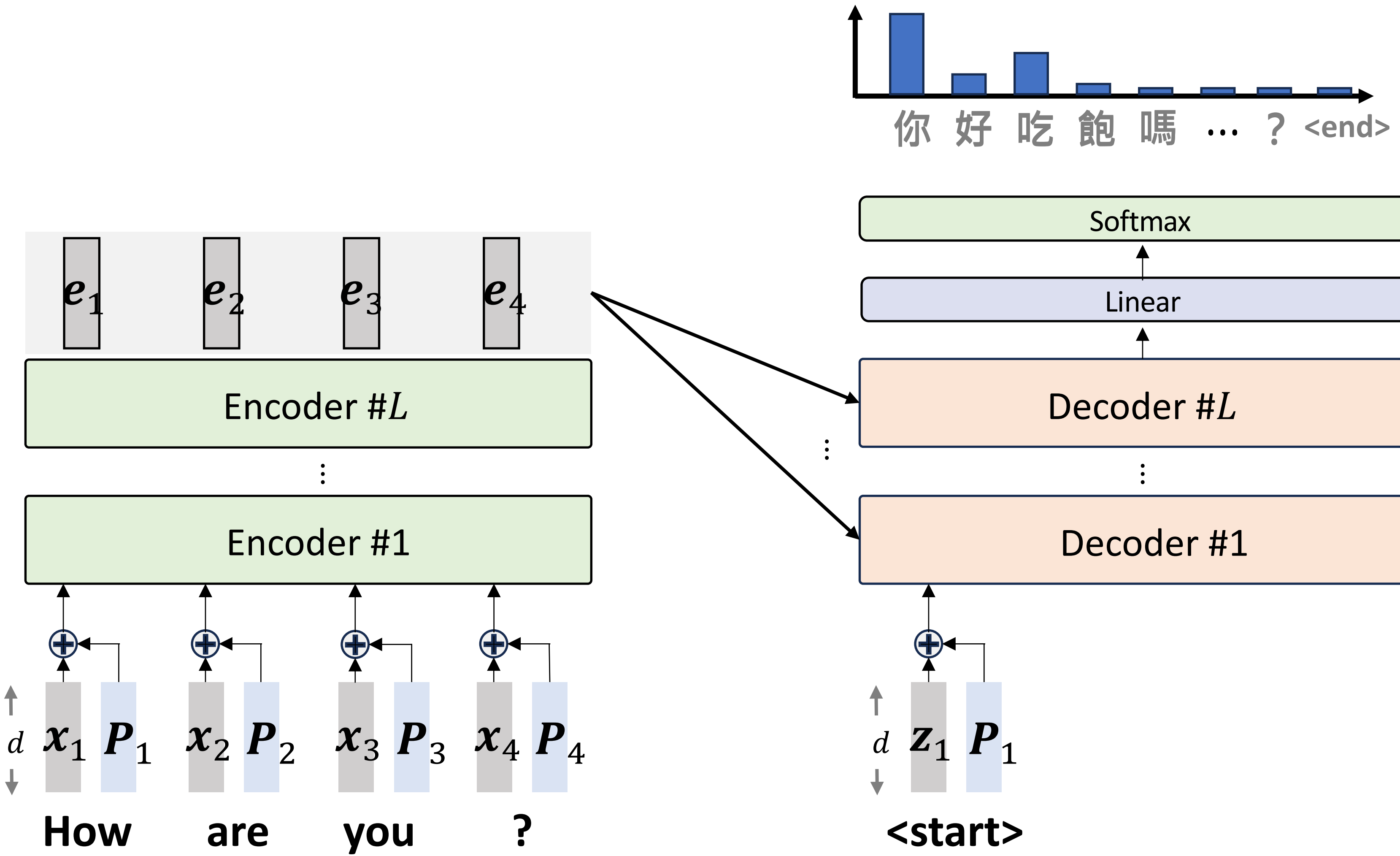
Encoder #6
Encoder #5
Encoder #4
Encoder #3
Encoder #2
Encoder #1
Embedded Tokens
d
x_1 P_1
x_2 P_2
x_3 P_3
x_4 P_4
x_5 P_5
Tokens
I
bought
an
apple
watch



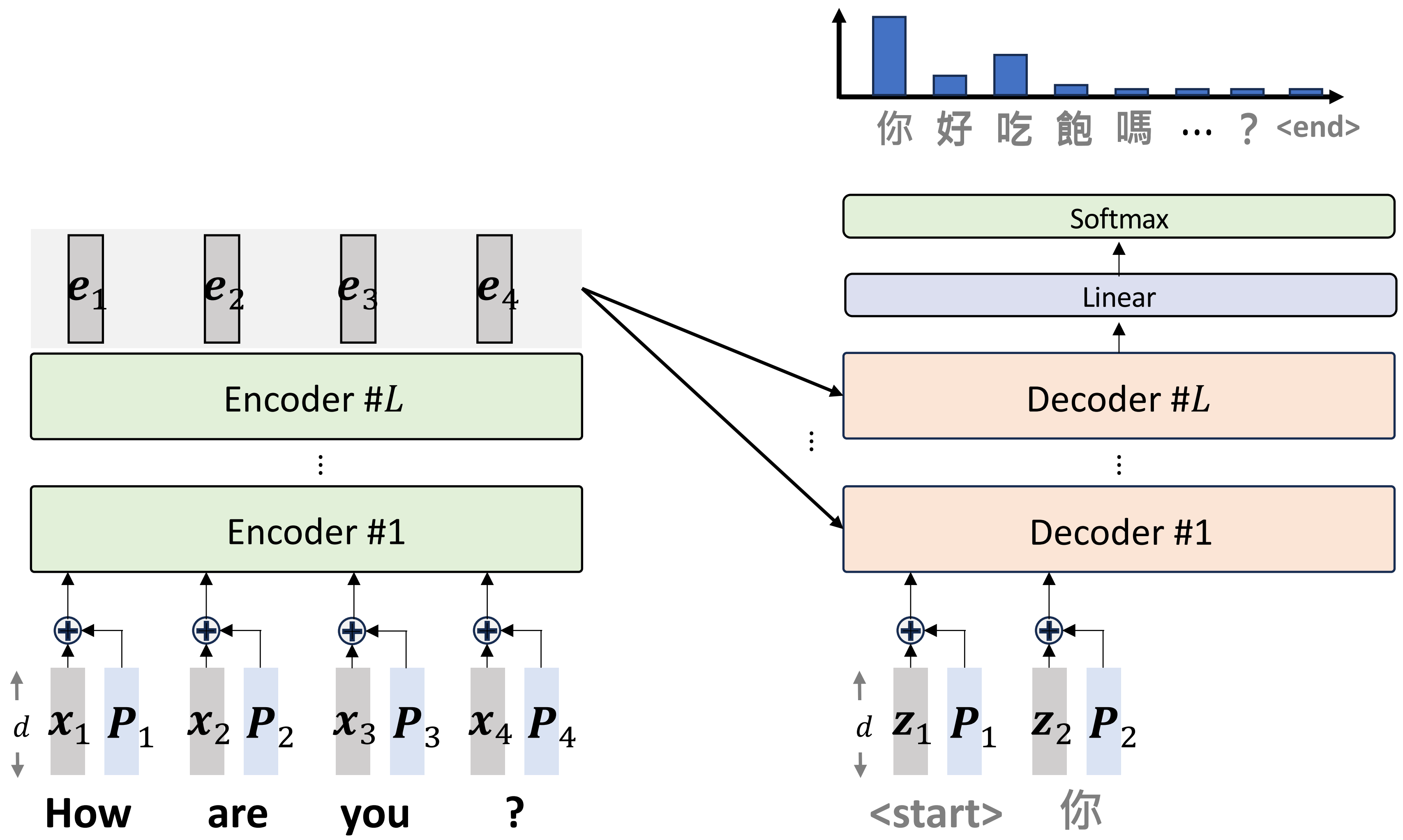
Encoder #6
Encoder #5
Encoder #4
Encoder #3
Encoder #2
Encoder #1
Embedded Tokens
d
x1 P1
x2 P2
x3 P3
x4 P4
Tokens
How
are
you
?



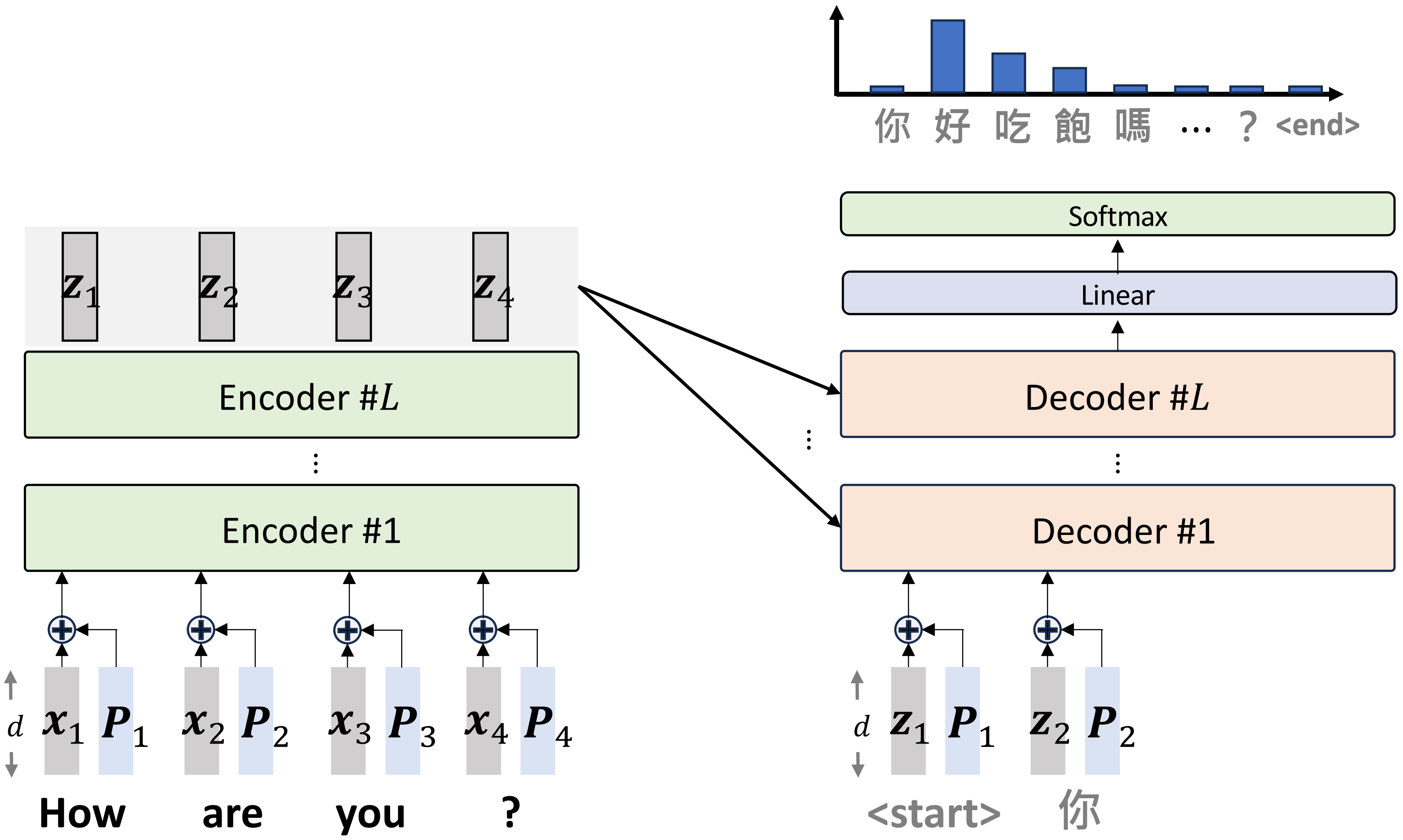
Encoder #6
⋮
Encoder #1
Embedded Tokens
d
$\boldsymbol{x}_1$ $\boldsymbol{P}_1$
$\boldsymbol{x}_2$ $\boldsymbol{P}_2$
$\boldsymbol{x}_3$ $\boldsymbol{P}_3$
$\boldsymbol{x}_4$ $\boldsymbol{P}_4$
Tokens
How
are
you
?



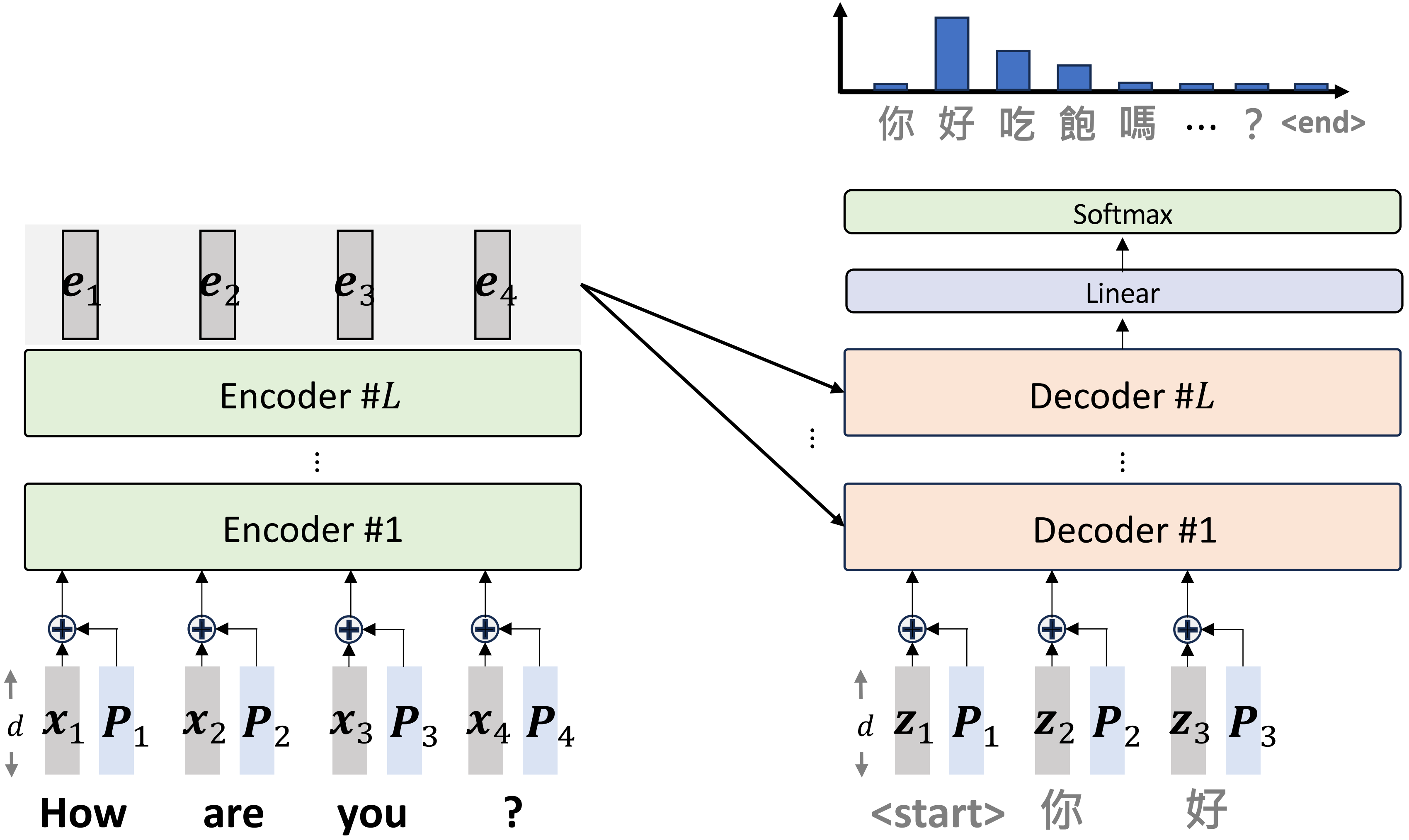
你 好 吃 飽 嗎 … ? <end>
Softmax
Linear
e_1 e_2 e_3 e_4
Encoder #L
Decoder #L
Encoder #1
Decoder #1
d x_1 P_1 x_2 P_2 x_3 P_3 x_4 P_4
d z_1 P_1
How are you ?
<start>

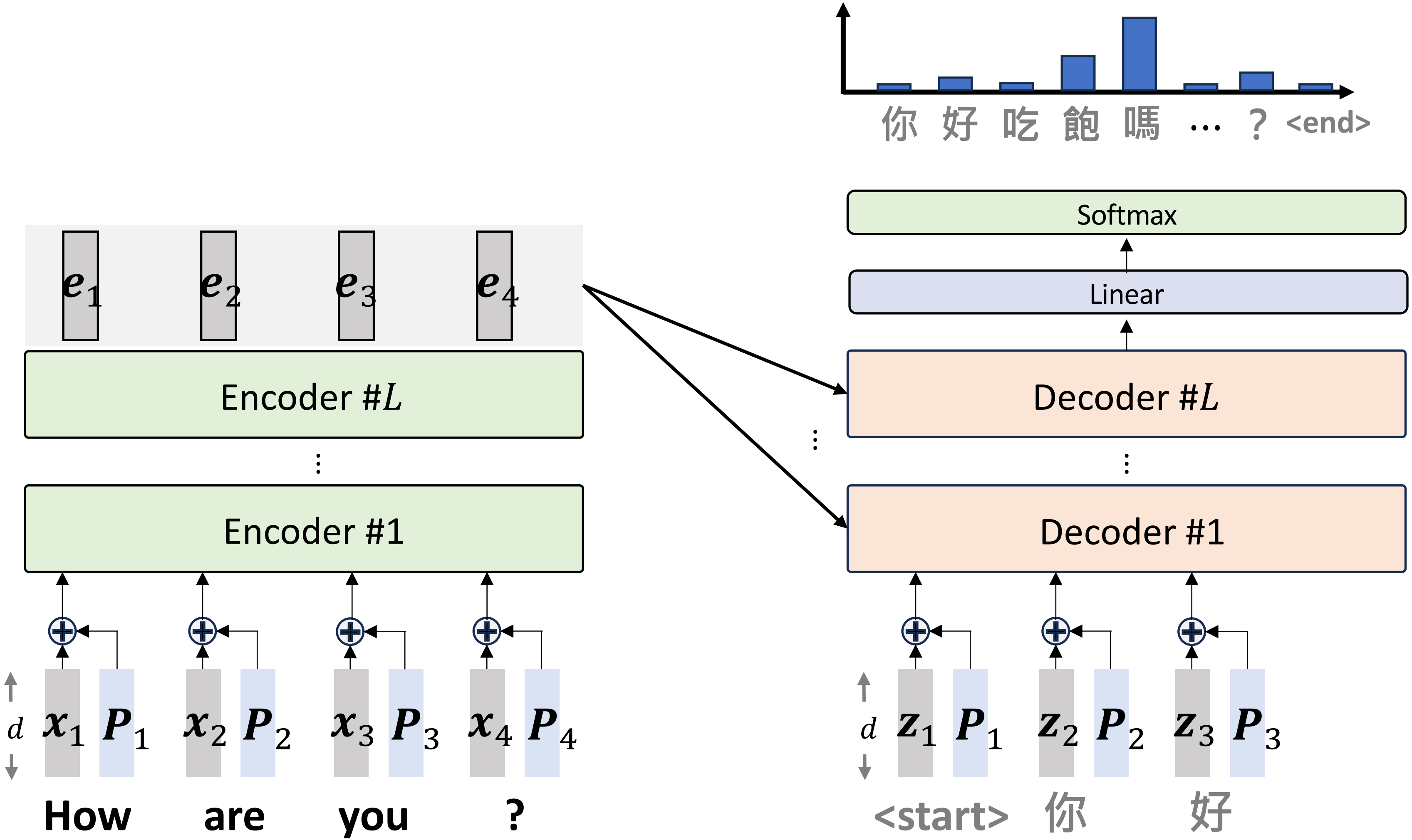


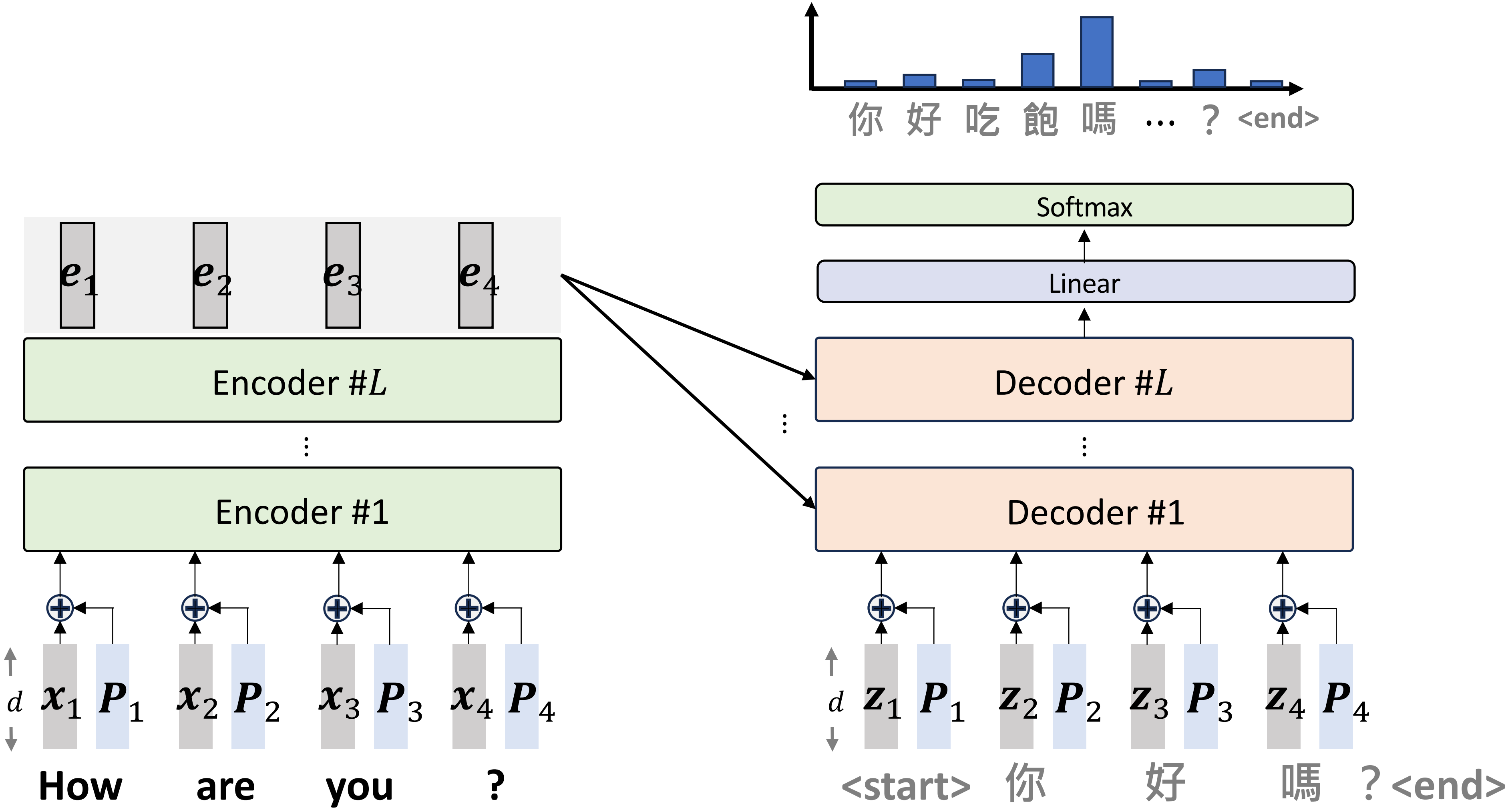
你 好 吃 飽 嗎 … ? <end>
Softmax
Linear
e_1 e_2 e_3 e_4
Encoder #L
Encoder #1
Decoder #L
Decoder #1
d
x_1 P_1 x_2 P_2 x_3 P_3 x_4 P_4
How are you ?
z_1 P_1 z_2 P_2
<start> 你



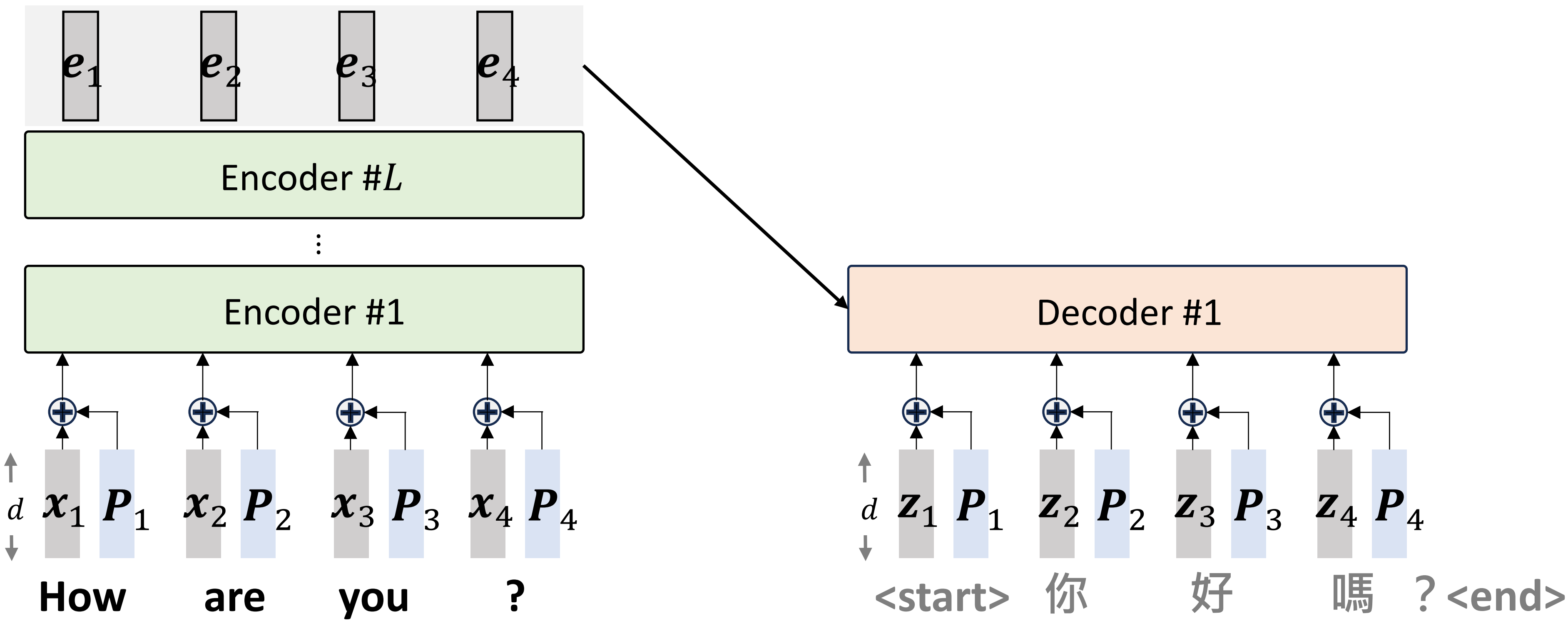
你 好 吃 飽 嗎 … ? <end>
Softmax
Linear
z_1 z_2 z_3 z_4
Encoder #L
Decoder #L
Encoder #1
Decoder #1
d x_1 P_1 x_2 P_2 x_3 P_3 x_4 P_4
How are you ?
d z_1 P_1 z_2 P_2
<start> 你


你 好 吃 飽 嗎 … ? <end>
Softmax
Linear
Decoder #L
Decoder #1
Encoder #L
Encoder #1
e1 e2 e3 e4
d x1 P1 x2 P2 x3 P3 x4 P4
How are you ?
d z1 P1 z2 P2 z3 P3
<start> 你 好

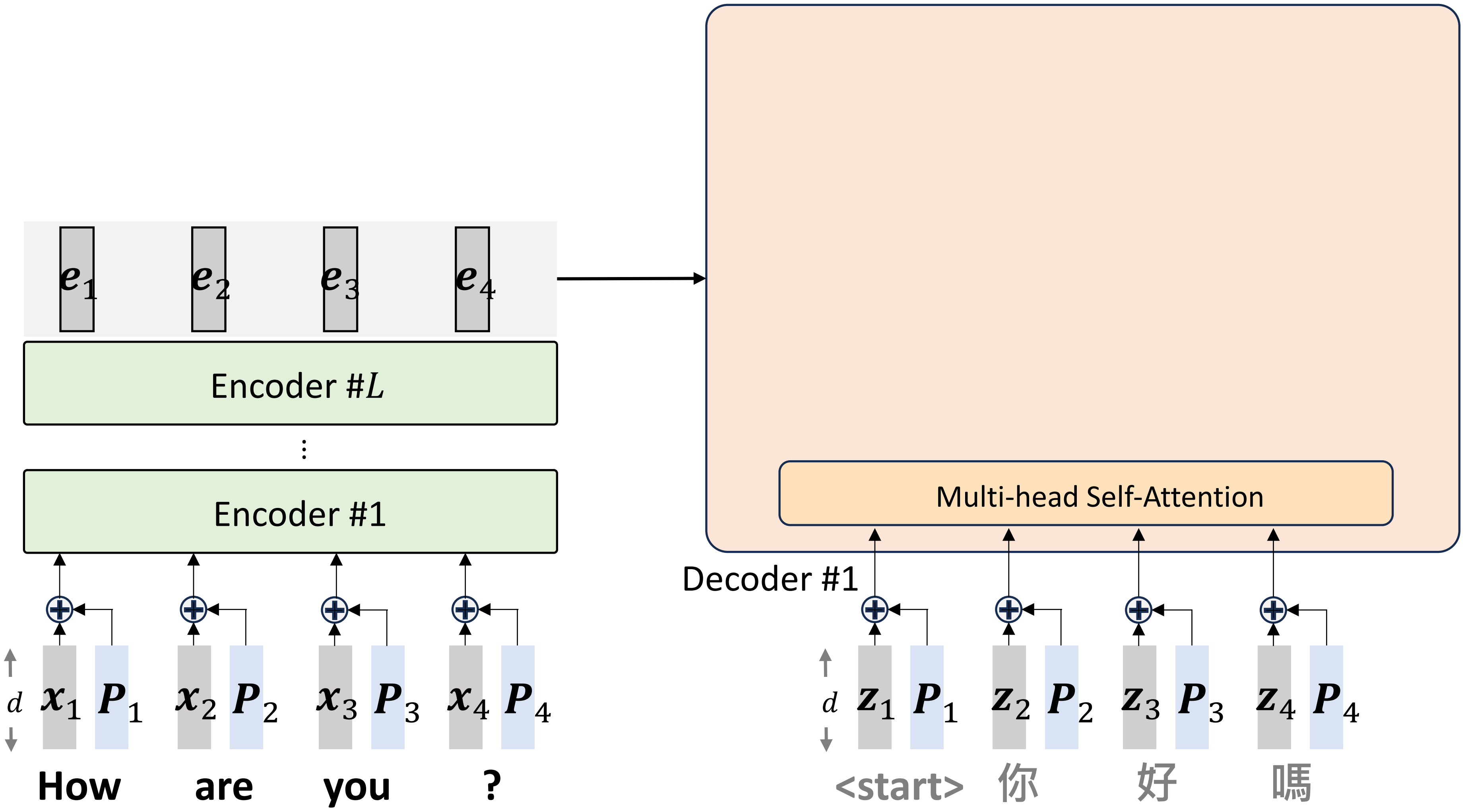

你 好 吃 飽 嗎 … ? <end>
Softmax
Linear
e_1 e_2 e_3 e_4
Encoder #L
Encoder #1
Decoder #L
Decoder #1
d x_1 P_1 x_2 P_2 x_3 P_3 x_4 P_4
d z_1 P_1 z_2 P_2 z_3 P_3
How are you ?
<start> 你 好



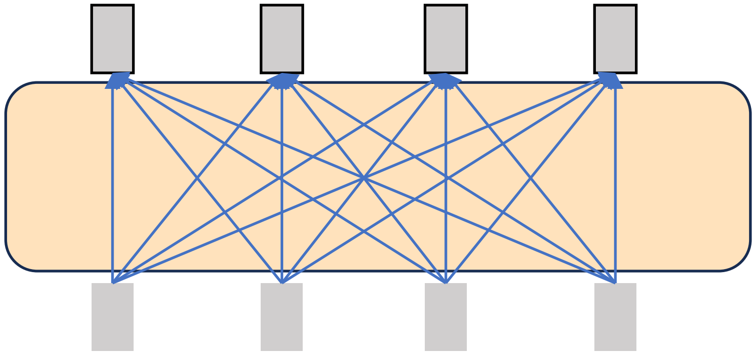
你 好 吃 飽 嗎 … ？ <end>
Softmax
Linear
e_1 e_2 e_3 e_4
Encoder #L
Encoder #1
Decoder #L
Decoder #1
d x_1 P_1 x_2 P_2 x_3 P_3 x_4 P_4
How are you ?
d z_1 P_1 z_2 P_2 z_3 P_3 z_4 P_4
<start> 你 好 嗎 ？<end>

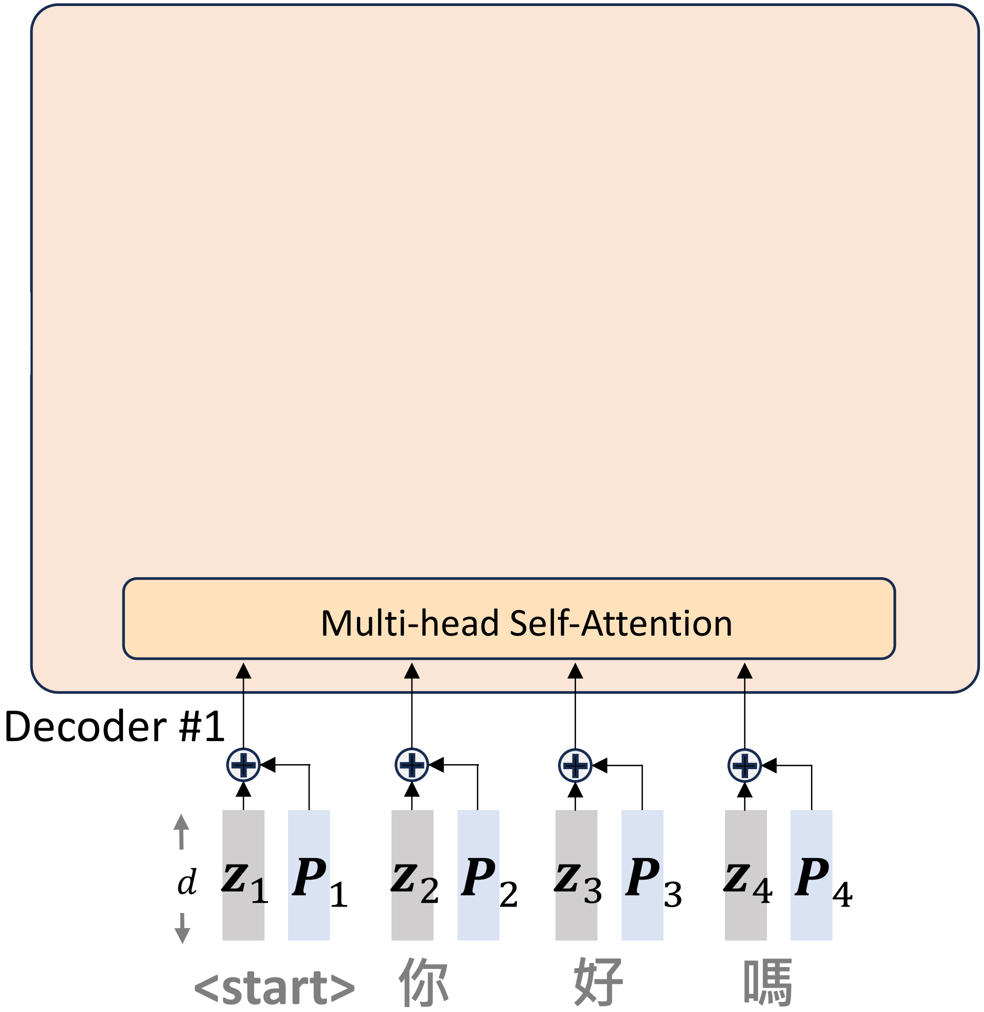


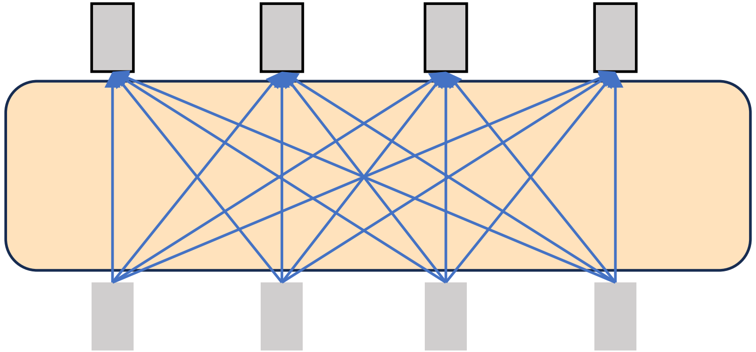
e_1
e_2
e_3
e_4
Encoder #L
Encoder #1
Decoder #1
d
x_1 P_1 x_2 P_2 x_3 P_3 x_4 P_4
How are you ?
d
z_1 P_1 z_2 P_2 z_3 P_3 z_4 P_4
<start> 你 好 嗎 ？<end>

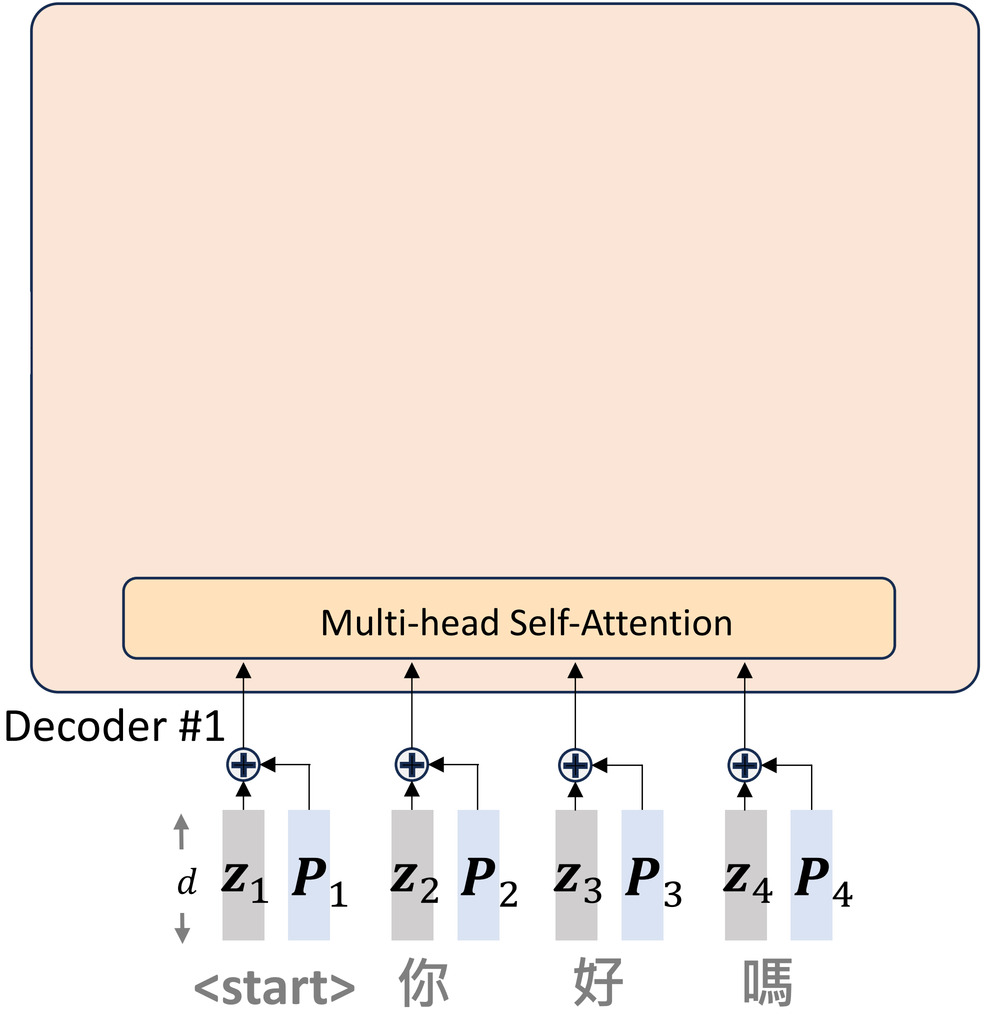


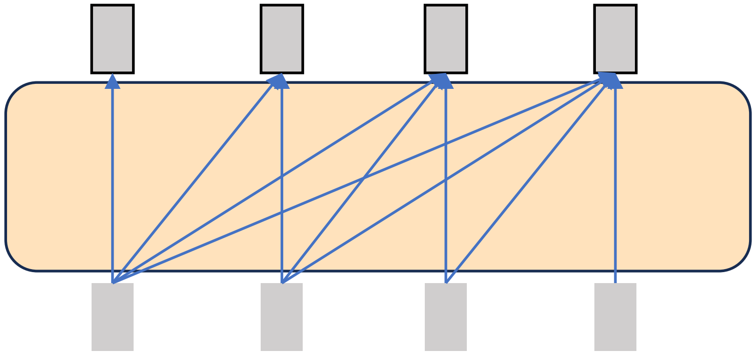
e_1 e_2 e_3 e_4
Encoder #L
Encoder #1
d x_1 P_1 x_2 P_2 x_3 P_3 x_4 P_4
How are you ?
Multi-head Self-Attention
Decoder #1
d z_1 P_1 z_2 P_2 z_3 P_3 z_4 P_4
<start> 你 好 嗎




Multi-head Self-Attention
Decoder #1
d
z_1 P_1 z_2 P_2 z_3 P_3 z_4 P_4
<start> 你 好 嗎



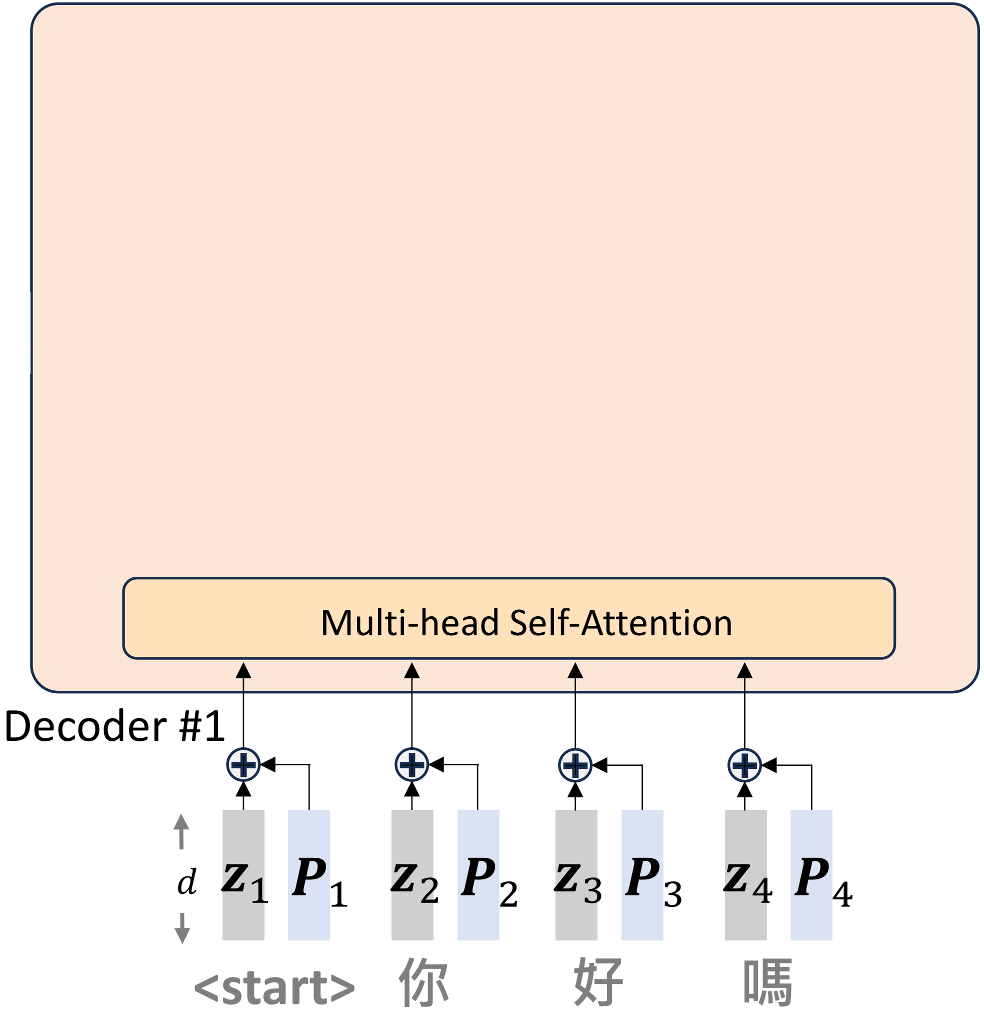

Multi-head Self-Attention
Decoder #1
d
z_1 P_1 z_2 P_2 z_3 P_3 z_4 P_4
<start> 你 好 嗎

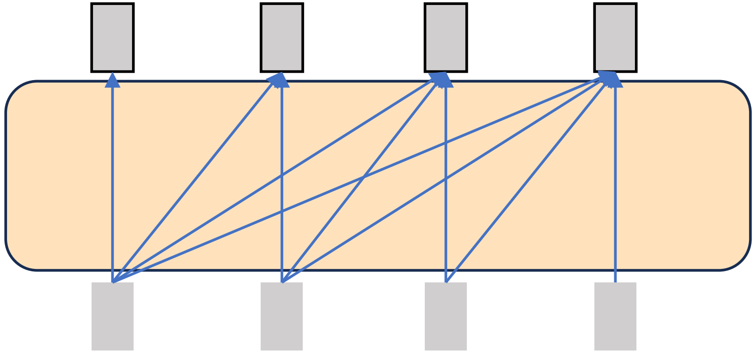


$$\text{Attention}(Q,K,V) = V\ \text{softmax}\left(\frac{K^\top Q}{\sqrt{d_k}}\right)$$

$$\text{MaskedAttention}(Q,K,V) = V\ \text{softmax}\left(\frac{K^\top Q}{\sqrt{d_k}} + M\right)$$

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -\infty & 0 & 0 & 0 & 0 \\ -\infty & -\infty & 0 & 0 & 0 \\ -\infty & -\infty & -\infty & 0 & 0 \\ -\infty & -\infty & -\infty & -\infty & 0 \end{bmatrix}$$

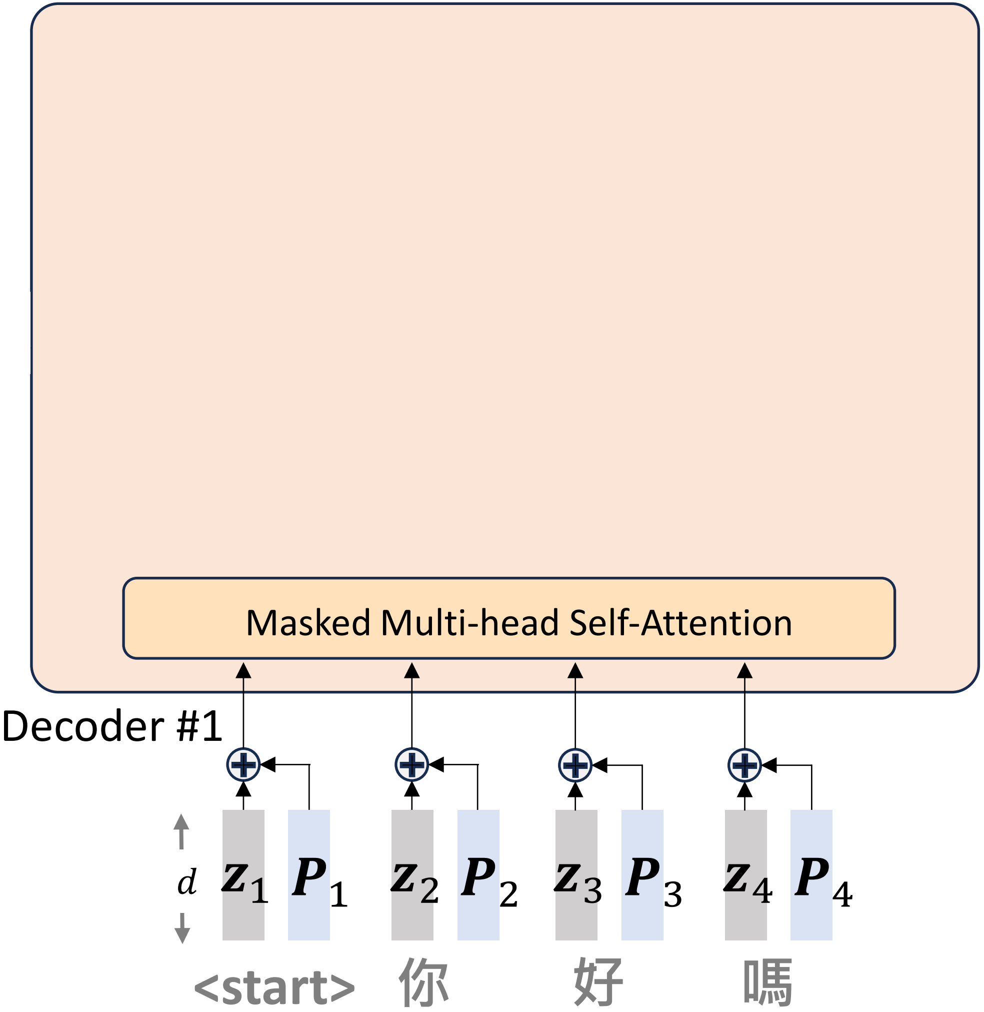


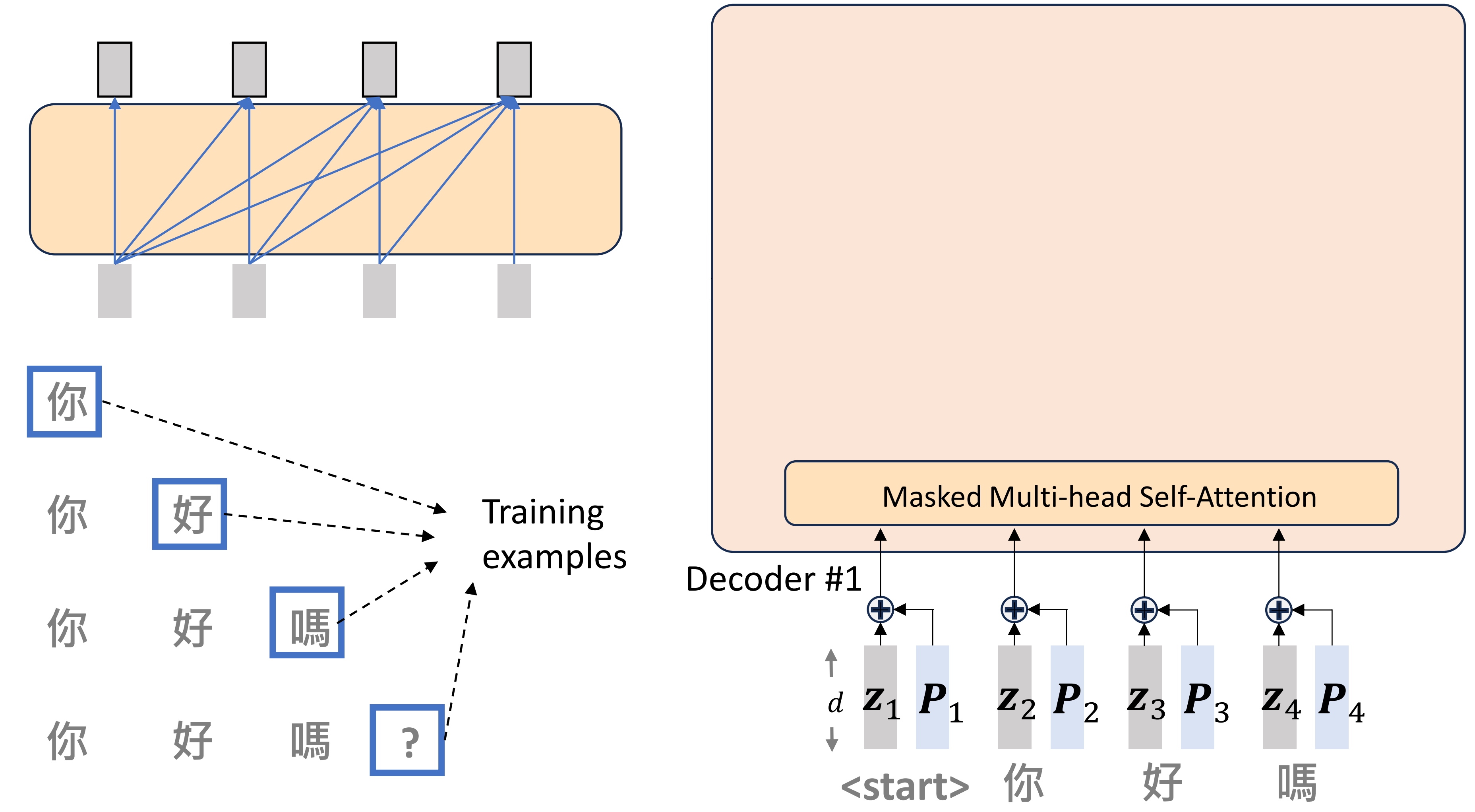


$$\text{Attention}(Q,K,V) = V\ \text{softmax}\left(\frac{K^\top Q}{\sqrt{d_k}}\right)$$

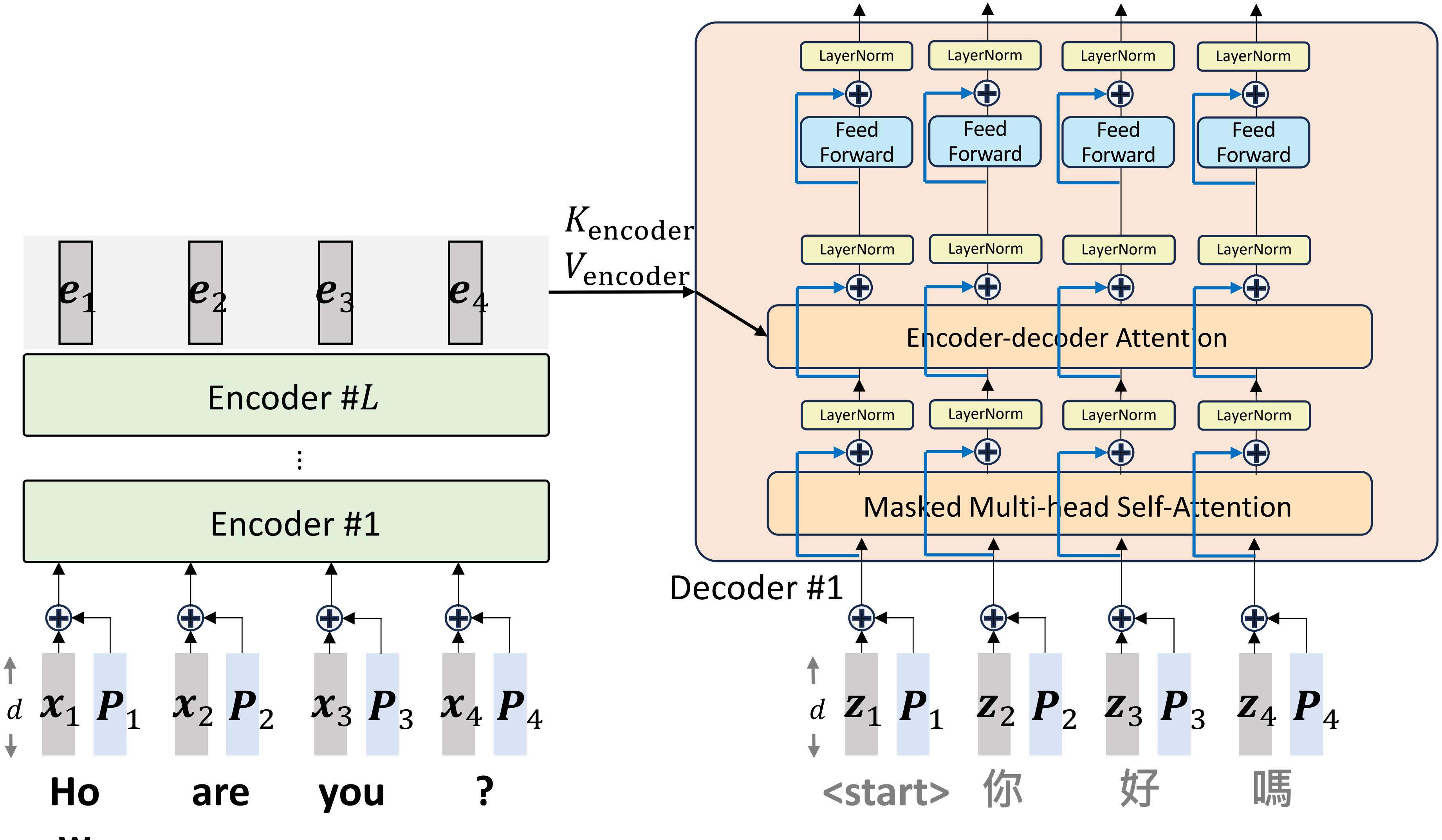
$$\text{MaskedAttention}(Q,K,V) = V\ \text{softmax}\left(\frac{K^\top Q}{\sqrt{d_k}} + M\right)$$

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -\infty & 0 & 0 & 0 & 0 \\ -\infty & -\infty & 0 & 0 & 0 \\ -\infty & -\infty & -\infty & 0 & 0 \\ -\infty & -\infty & -\infty & -\infty & 0 \end{bmatrix}$$

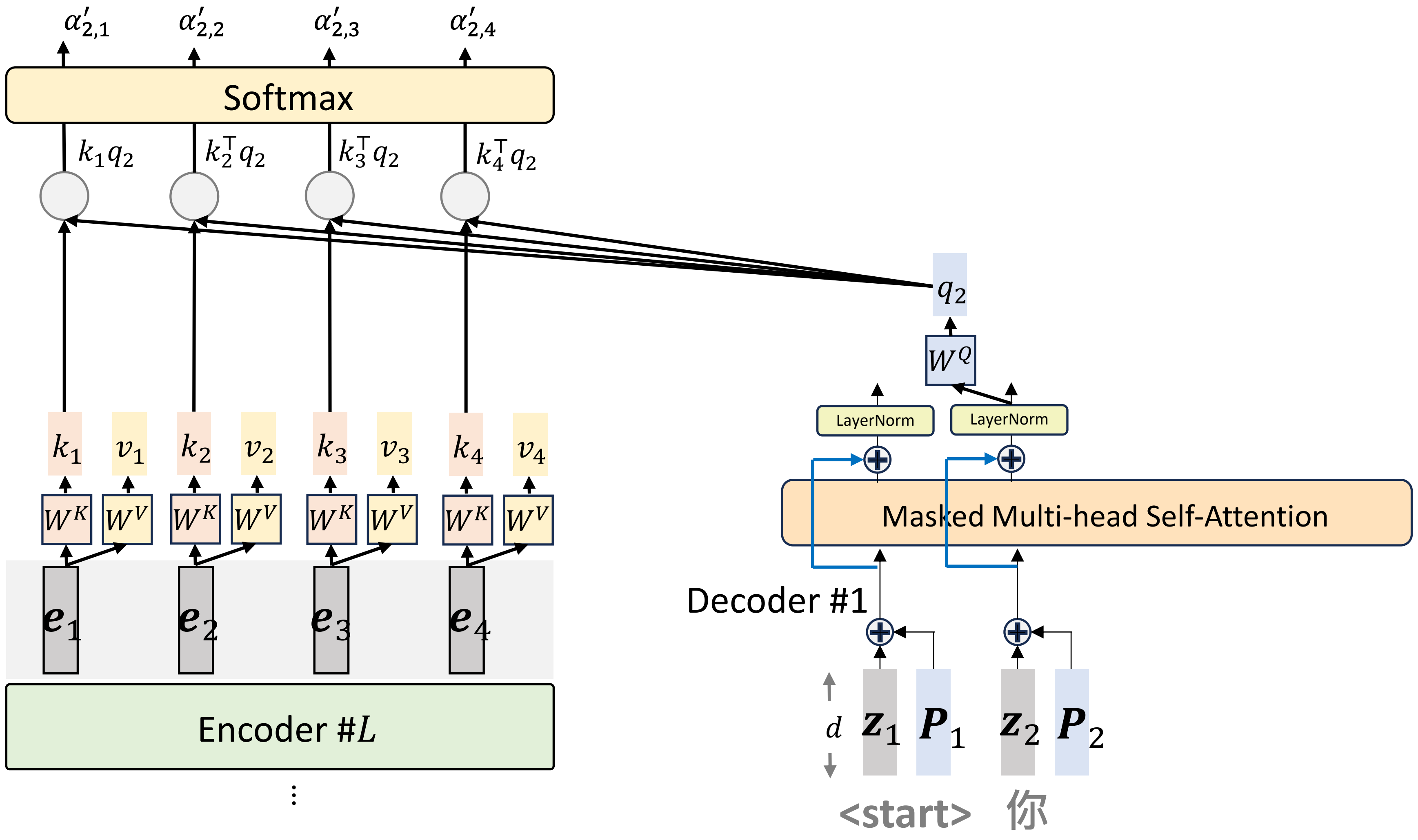




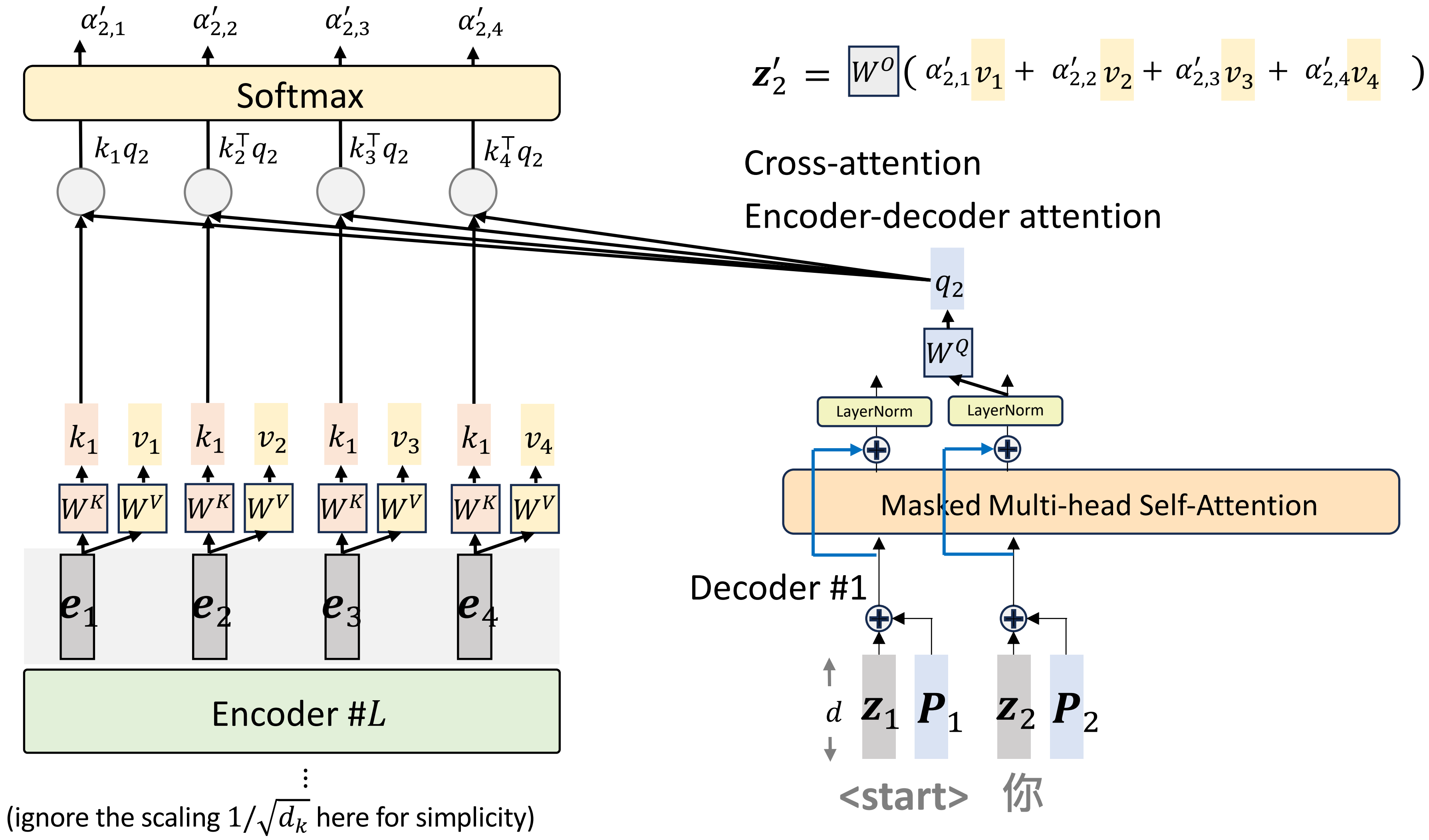
你
你 好
你 好 嗎
你 好 嗎 ?
Training examples
Masked Multi-head Self-Attention
Decoder #1
d
z1 P1 z2 P2 z3 P3 z4 P4
<start> 你 好 嗎



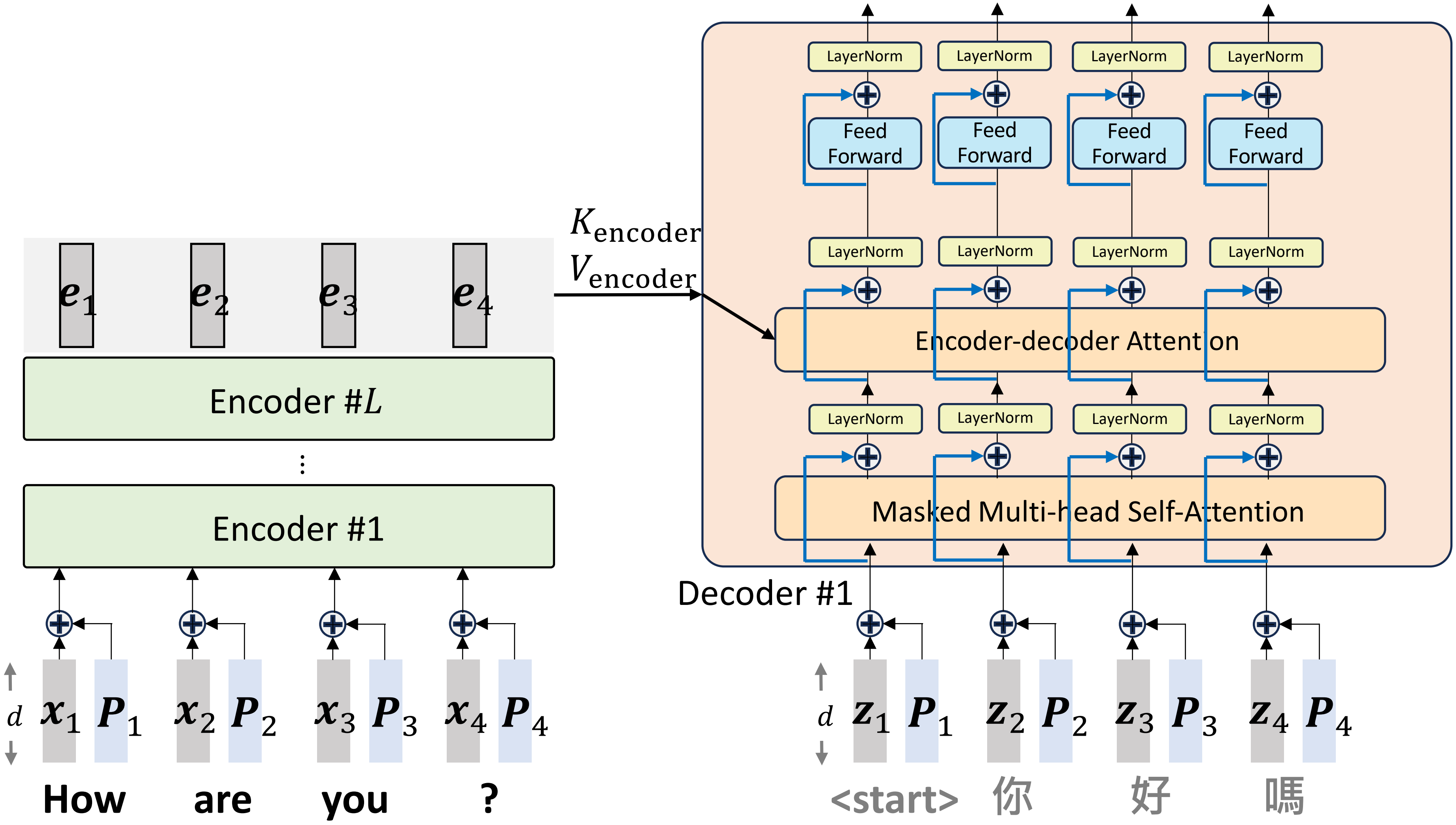
e_1
e_2
e_3
e_4
K_{encoder}
V_{encoder}
Encoder #L
Encoder #1
LayerNorm
Feed Forward
LayerNorm
Encoder-decoder Attention
LayerNorm
Masked Multi-head Self-Attention
Decoder #1
d
x_1 P_1 x_2 P_2 x_3 P_3 x_4 P_4
Ho
are
you
?
d
z_1 P_1 z_2 P_2 z_3 P_3 z_4 P_4
<start>
你
好
嗎



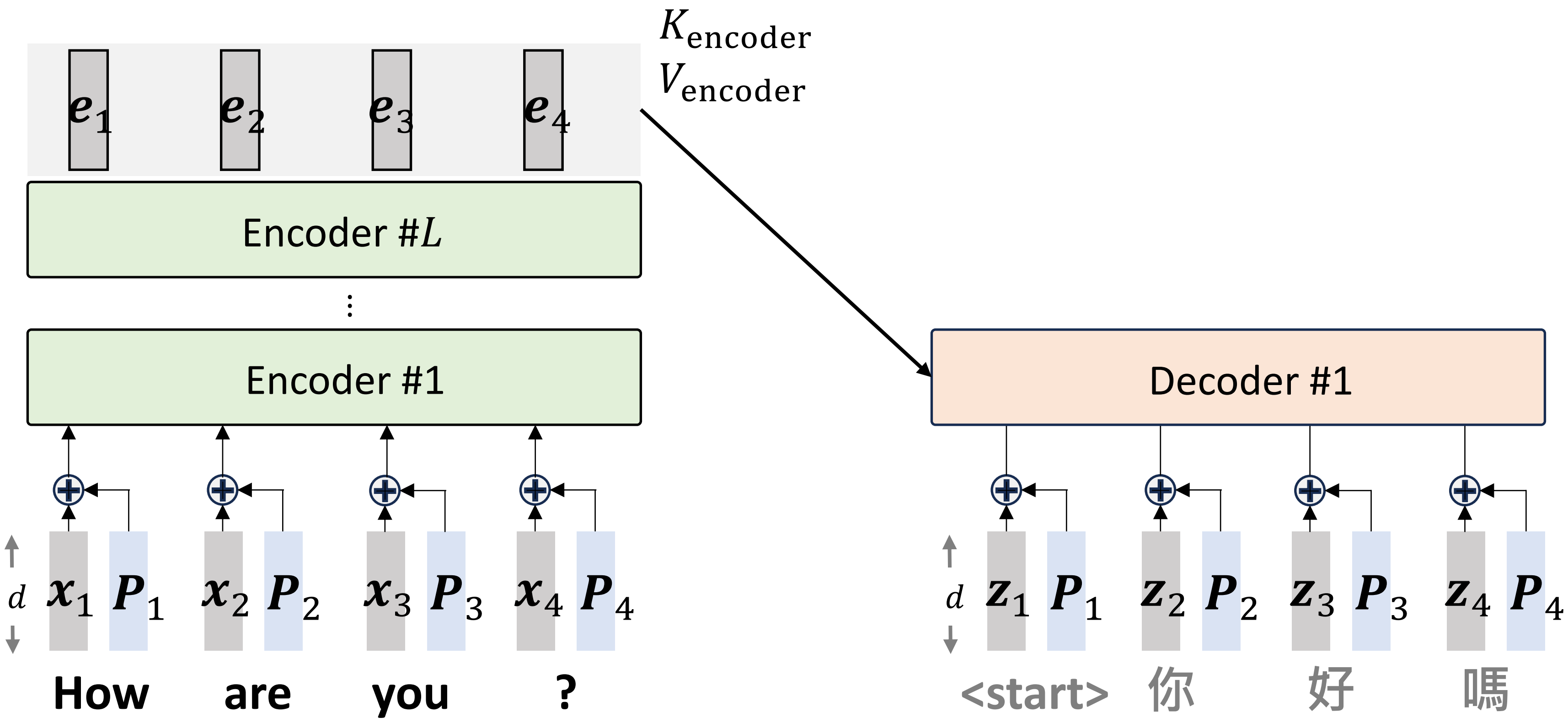
$\alpha'_{2,1}$
$\alpha'_{2,2}$
$\alpha'_{2,3}$
$\alpha'_{2,4}$
Softmax
$k_1 q_2$
$k_2^\top q_2$
$k_3^\top q_2$
$k_4^\top q_2$
q_2
W^Q
LayerNorm
LayerNorm
k_1 v_1 k_2 v_2 k_3 v_3 k_4 v_4
W^K W^V W^K W^V W^K W^V W^K W^V
Masked Multi-head Self-Attention
$\boldsymbol{e}_1$ $\boldsymbol{e}_2$ $\boldsymbol{e}_3$ $\boldsymbol{e}_4$
Decoder #1
Encoder #L
d
$\boldsymbol{z}_1$ $\boldsymbol{P}_1$ $\boldsymbol{z}_2$ $\boldsymbol{P}_2$
<start>
你



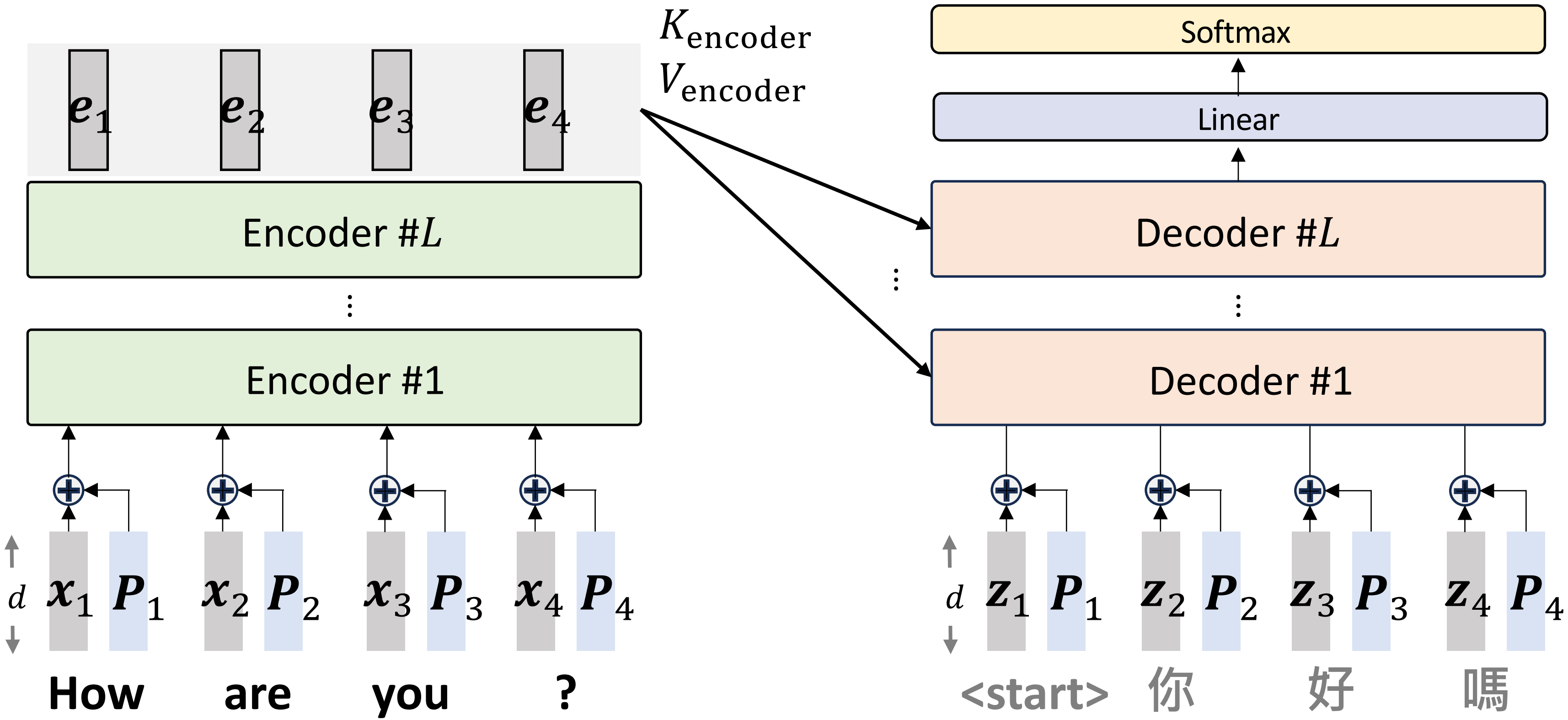
$z'_2 = \boxed{W^O}(\alpha'_{2,1} v_1 + \alpha'_{2,2} v_2 + \alpha'_{2,3} v_3 + \alpha'_{2,4} v_4)$
Cross-attention
Encoder-decoder attention
$\alpha'_{2,1}$ $\alpha'_{2,2}$ $\alpha'_{2,3}$ $\alpha'_{2,4}$
Softmax
$k_1 q_2$ $k_2^\top q_2$ $k_3^\top q_2$ $k_4^\top q_2$
q_2
W^Q
LayerNorm
LayerNorm
Masked Multi-head Self-Attention
Decoder #1
k_1 v_1 k_1 v_2 k_1 v_3 k_1 v_4
W^K W^V W^K W^V W^K W^V W^K W^V
e_1 e_2 e_3 e_4
Encoder #L
d
z_1 P_1 z_2 P_2
<start> 你
(ignore the scaling $1/\sqrt{d_k}$ here for simplicity)



LayerNorm
Feed Forward
LayerNorm
Encoder-decoder Attention
LayerNorm
Masked Multi-head Self-Attention
Decoder #1
K_{encoder}
V_{encoder}
e_1 e_2 e_3 e_4
Encoder #L
⋮
Encoder #1
d
x_1 P_1 x_2 P_2 x_3 P_3 x_4 P_4
How are you ?
z_1 P_1 z_2 P_2 z_3 P_3 z_4 P_4
<start> 你 好 嗎



K_{encoder}
V_{encoder}
$\boldsymbol{e}_1$ $\boldsymbol{e}_2$ $\boldsymbol{e}_3$ $\boldsymbol{e}_4$
Encoder #L
Encoder #1
Decoder #1
d
$\boldsymbol{x}_1$ $\boldsymbol{P}_1$ $\boldsymbol{x}_2$ $\boldsymbol{P}_2$ $\boldsymbol{x}_3$ $\boldsymbol{P}_3$ $\boldsymbol{x}_4$ $\boldsymbol{P}_4$
How are you ?
$\boldsymbol{z}_1$ $\boldsymbol{P}_1$ $\boldsymbol{z}_2$ $\boldsymbol{P}_2$ $\boldsymbol{z}_3$ $\boldsymbol{P}_3$ $\boldsymbol{z}_4$ $\boldsymbol{P}_4$
<start> 你 好 嗎



K_{encoder}
V_{encoder}
Softmax
Linear
$\boldsymbol{e}_1$ $\boldsymbol{e}_2$ $\boldsymbol{e}_3$ $\boldsymbol{e}_4$
Encoder #L
Encoder #1
Decoder #L
Decoder #1
d $\boldsymbol{x}_1$ $\boldsymbol{P}_1$ $\boldsymbol{x}_2$ $\boldsymbol{P}_2$ $\boldsymbol{x}_3$ $\boldsymbol{P}_3$ $\boldsymbol{x}_4$ $\boldsymbol{P}_4$
d $\boldsymbol{z}_1$ $\boldsymbol{P}_1$ $\boldsymbol{z}_2$ $\boldsymbol{P}_2$ $\boldsymbol{z}_3$ $\boldsymbol{P}_3$ $\boldsymbol{z}_4$ $\boldsymbol{P}_4$
How are you ?
<start> 你 好 嗎

Transformer & Multi-Head Attention

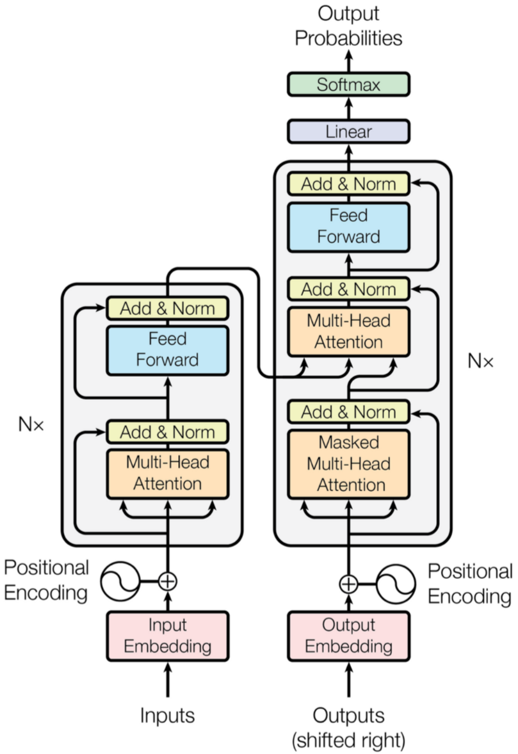


Figure 1: The Transformer - model architecture.

"Attention Is All You Need"
https://arxiv.org/abs/1706.03762

Summary: Attention and Transformers

- Attention weights used to compute the context vector, which is a weighted sum of the input at different positions
- Context vector is used to update the hidden state of the model, which is used to generate the final output
- "Pay attention" to different parts of the input, depending on the task at hand → more accurate and natural-sounding output, esp. when working with longer inputs (e.g. paragraphs)

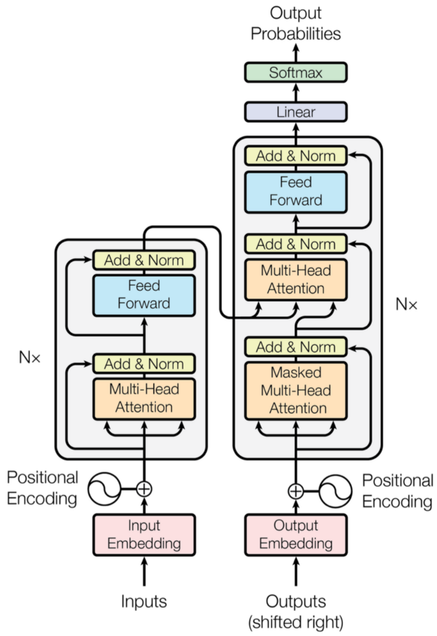


Figure 1: The Transformer - model architecture.

Ways Attention was used in the *original* Transformer Architecture

- **Encoder-decoder cross-attention**
  - Allow decoder layers to attend all parts of the latent representation produced by the encoder
  - Pull context from the encoder sequence over to the decoder

- **Self-attention in the encoder**
  - Allow the model to attend to all positions in the previous encoder layer
  - Embeds context about how elements in the sequence relate to one another

- **Masked self-attention in the decoder**
  - Allow the model to attend to all positions in the previous decoder layer up to and including the current position (during auto-regressive process)
  - Prevent forward looking bias by stopping leftward information flow during training
  - Also embed context about how elements in the sequence relate to one another

* A. Vaswani *et al.*, "Attention is All you Need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

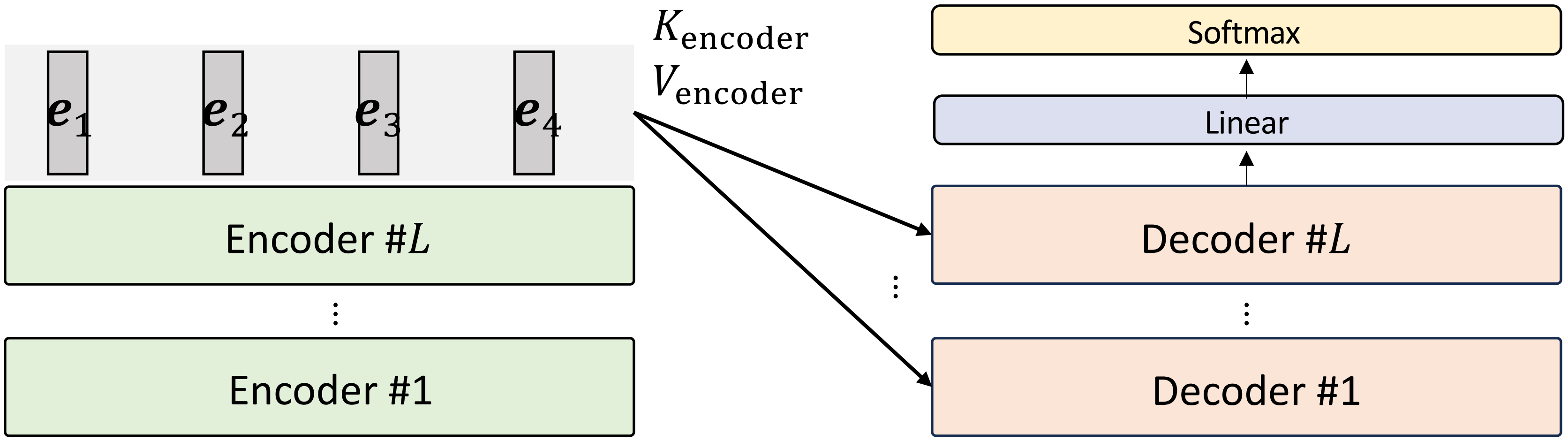
Encoder-Decoder Transformer

Examples:

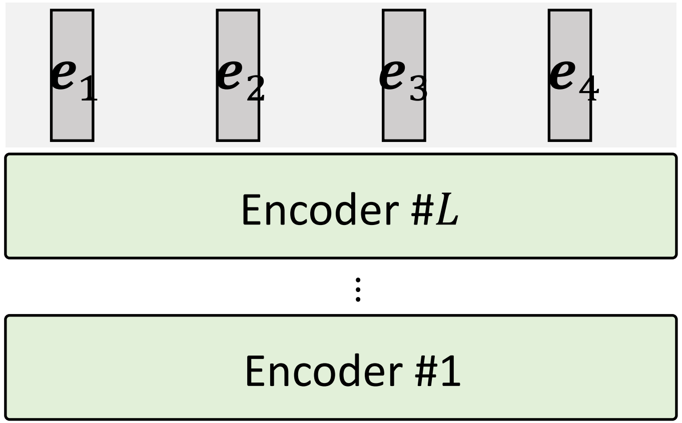
Attention is all you need, T5, BART.

Good for:

Machine translation, summarization. QA
(when input/target are sufficiently different)



Encoder-Decoder Transformer



Examples:

Attention is all you need, T5, BART.

Good for:

Machine translation, summarization. QA (when input/target are sufficiently different)

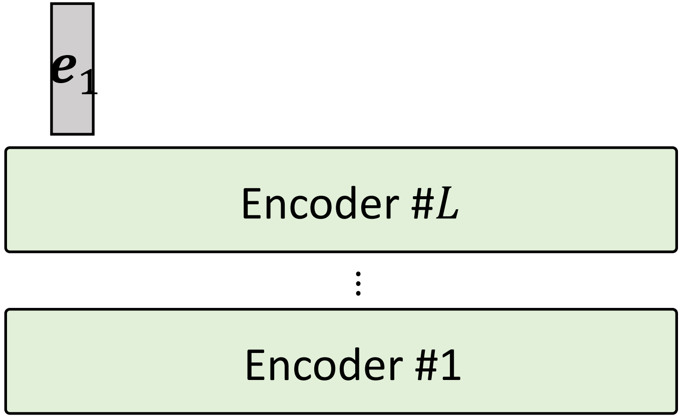
Encoder-only Transformer

Examples:

BERT, RoBERTa, DeBERTa, X-BERT

Good for:

Classification, sequence tagging, sentiment analysis
(Understand text, but not generate them)



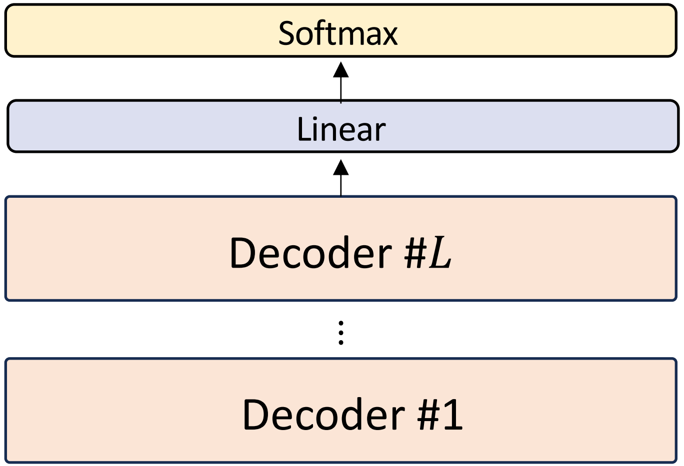
Decoder-only Transformer

Examples:

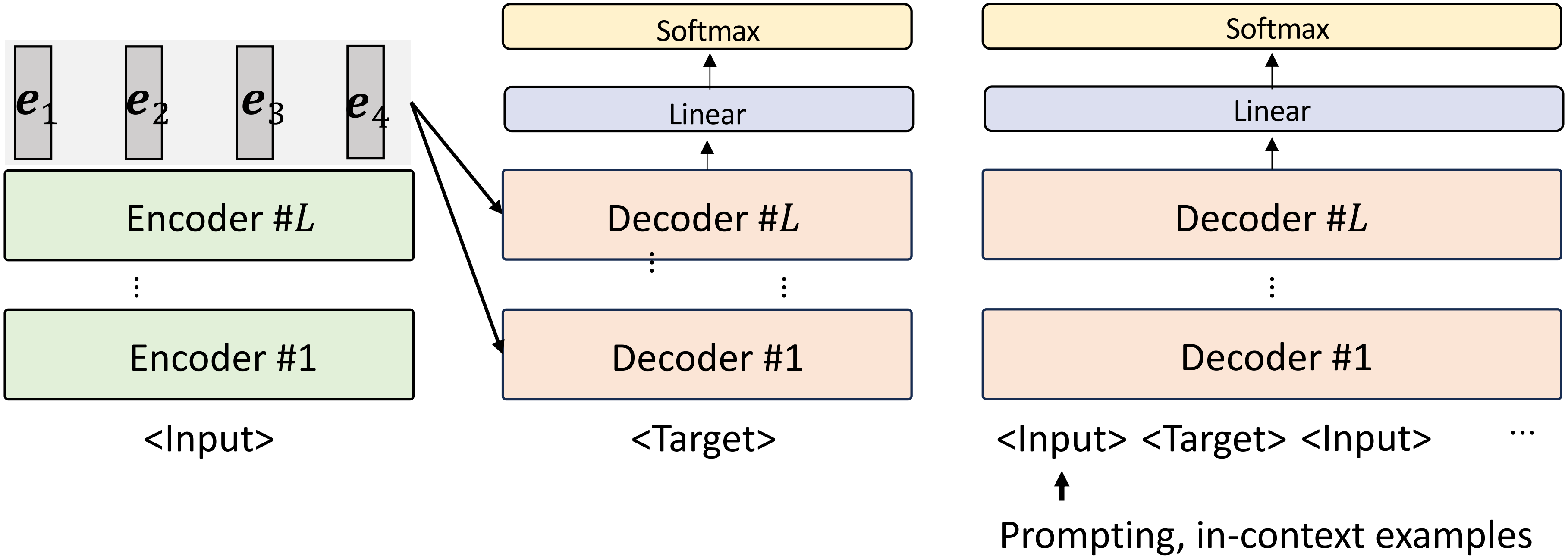
GPT-X (OpenAI), PaLM (Google), LLaMA (Meta)
BLOOM (BigScience)

Good for:

Text generation, multi-round conversation



Encoder-Decoder Transformer   Decoder-only Transformer



Different parameters for encoder/decoder   **Shared** parameters

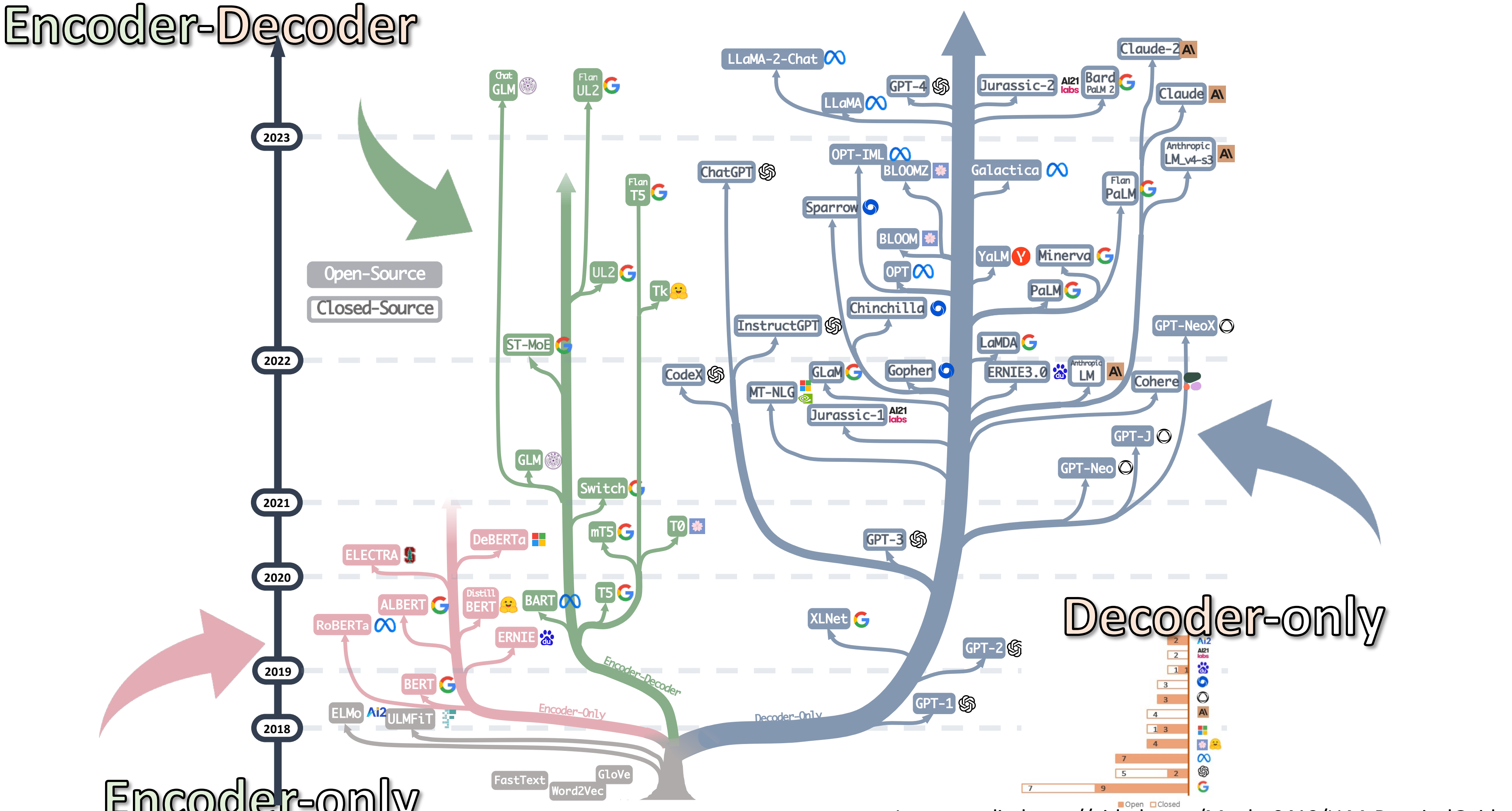
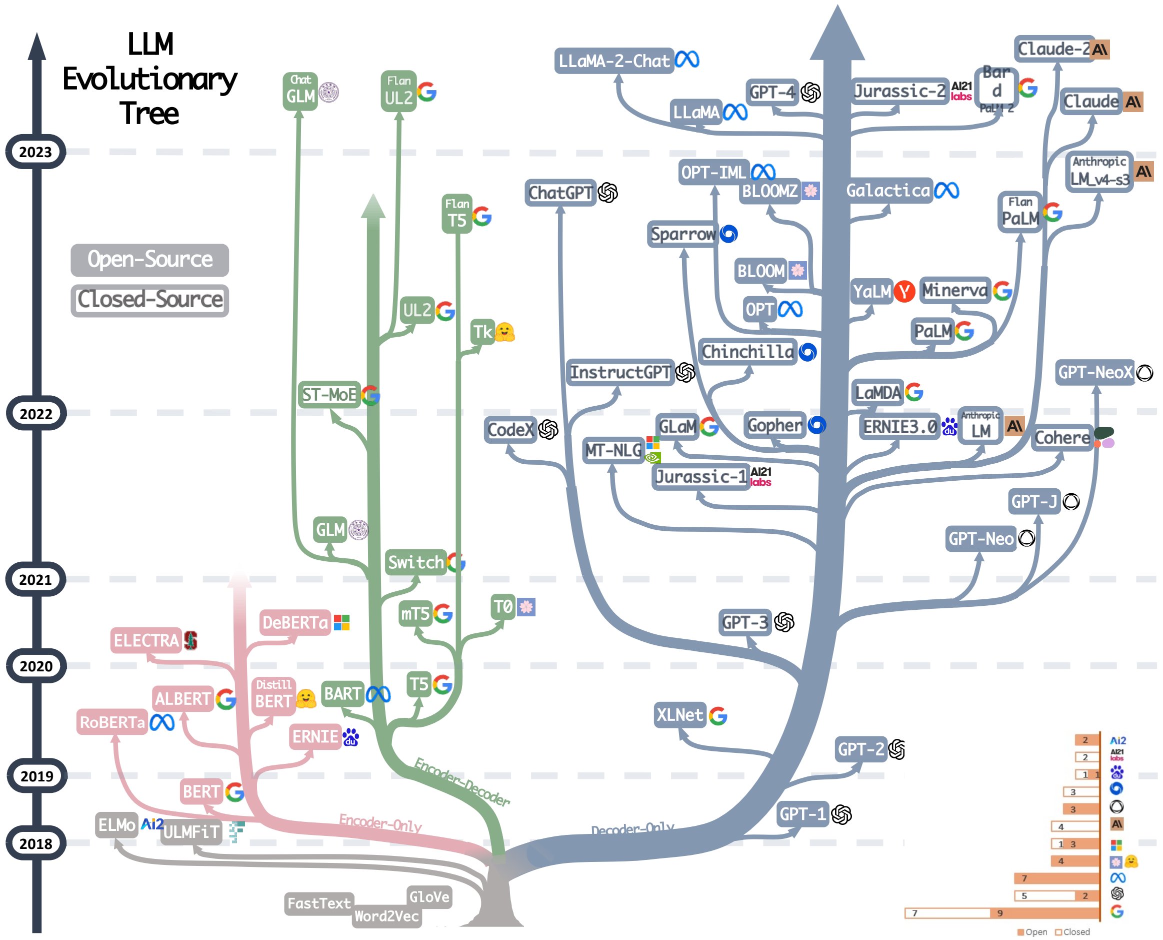


Image credit: https://github.com/Mooler0410/LLMsPracticalGuide


LLM Evolutionary Tree
2023
2022
2021
2020
2019
2018
Open-Source
Closed-Source
ChatGLM
Flan UL2
Flan T5
UL2
Tk
ST-MoE
GLM
Switch
mT5
T0
T5
BART
DeBERTa
ELECTRA
ALBERT
DistillBERT
RoBERTa
ERNIE
BERT
ELMo
ULMFiT
FastText
GloVe
Word2Vec
Encoder-Decoder
Encoder-Only
Decoder-Only
LLaMA-2-Chat
LLaMA
GPT-4
Jurassic-2
Bard
PaLM 2
Claude-2
Claude
Anthropic LM v4-s3
OPT-IML
BLOOMZ
Galactica
ChatGPT
Sparrow
BLOOM
OPT
YaLM
Minerva
Flan PaLM
PaLM
Chinchilla
InstructGPT
LaMDA
GPT-NeoX
CodeX
GLaM
Gopher
ERNIE3.0
Anthropic LM
Cohere
MT-NLG
Jurassic-1
GPT-J
GPT-Neo
GPT-3
XLNet
GPT-2
GPT-1
Open
Closed

Transformers vs. RNNs

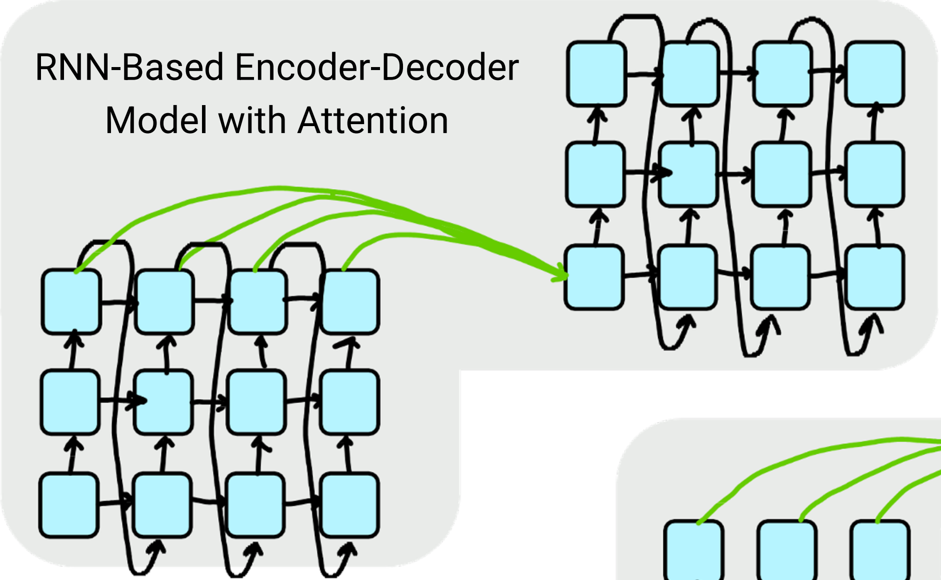
Challenges with RNNs	Transformers
• Long range dependencies • Gradient vanishing and explosion • Large # of training steps • Sequential/recurrence → can't parallelize • Complexity per layer: $O(n*d^2)$	• Can model long-range dependencies • No gradient vanishing and explosion • Fewer training steps • Can parallelize computation! • Complexity per layer: $O(n^2*d)$

- When sequence length (n) << representation dimension (d), the complexity per layer is lower for a Transformer model compared to RNN models ; no true for real-world LLMs

Differences in Attention Mechanism of RNN vs. Transformer

Feature	RNN with Attention (Bahdanau et al. 2015)	Transformer
Attention Type	Additive (Bahdanau) Attention	Scaled Dot Product Attention
Alignment	Based on decoder hidden state and encoder hidden states	Based on dot-product of query and keys (global attention)
Efficiency	Processes sequences step-by-step	Parallel processing of all positions
Context	Weighted sum of encoder hidden states at each step	Attends to all encoder positions for every output
Self-Attention	Not used	Self-attention in both encoder and decoder

Computational Dependencies for Recurrence vs. Attention



Transformer Advantages:

- # unparallelizable operations does not increase with sequence length.
- Each word interacts with each other, so maximum interaction distance is O(1).