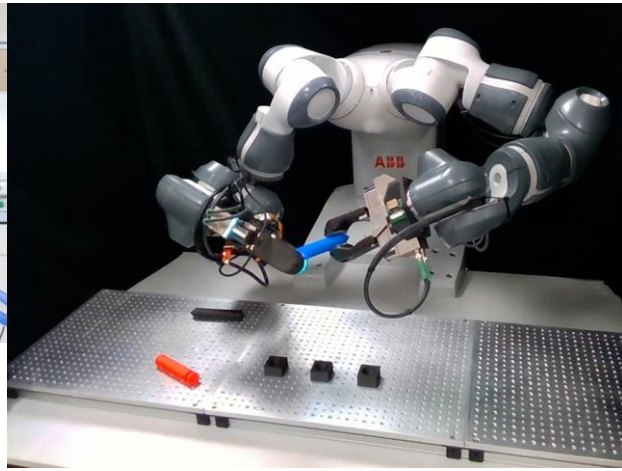
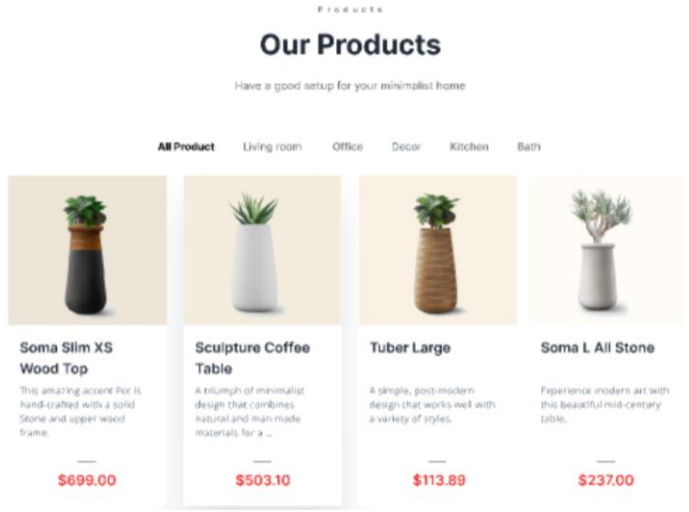
Offline RL

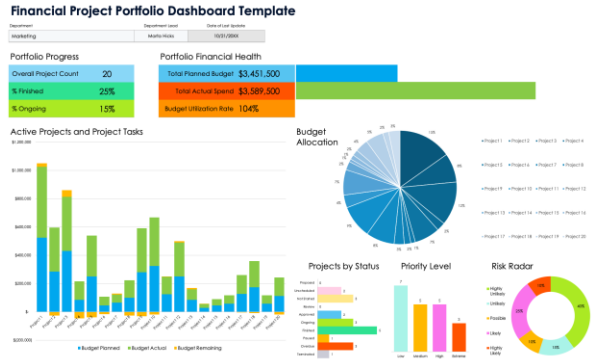
The problem of Offline RL

Applications

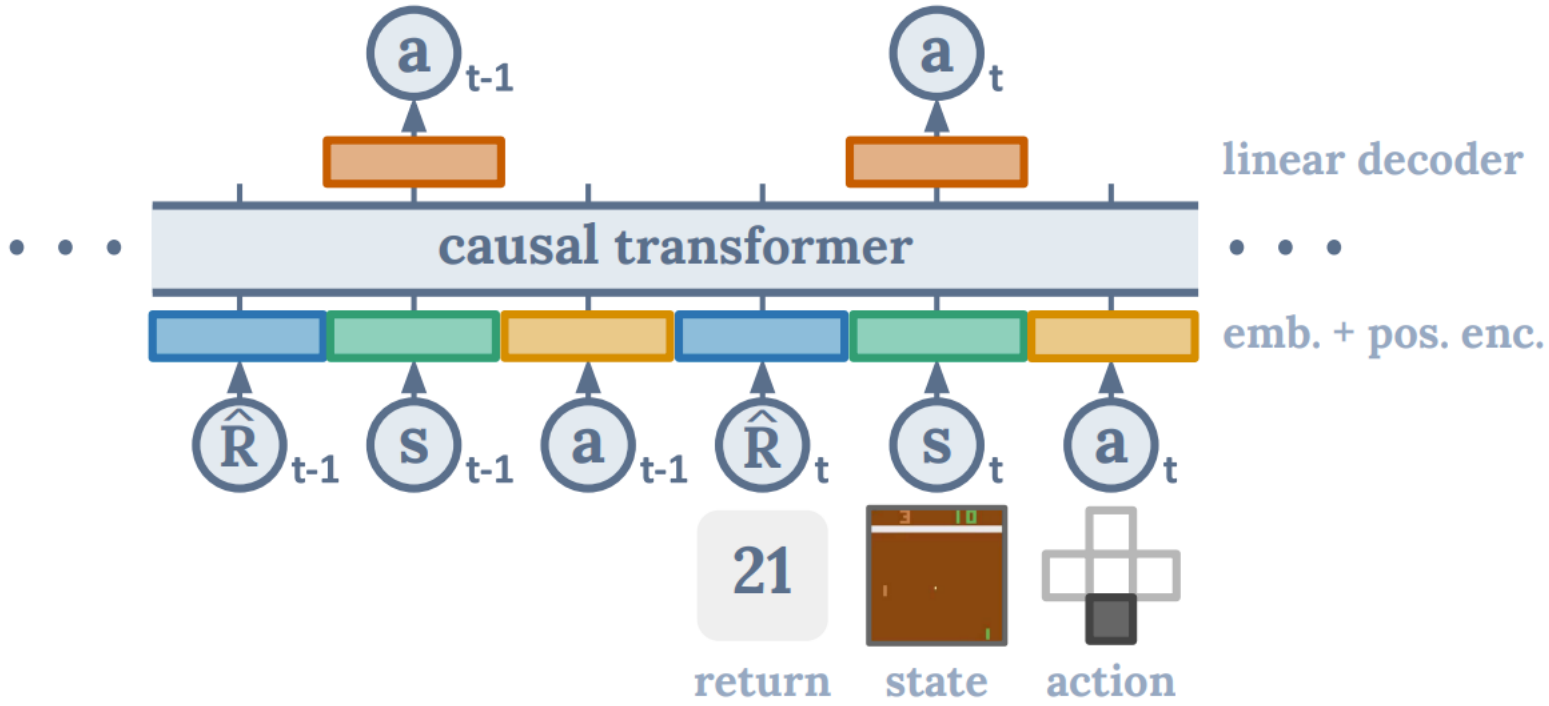


- Healthcare and Personalized Medicine
- Autonomous Driving
- Logistics and warehouse operations
- Robotics (Manipulation & Assistive Robots)
- Recommender Systems
- Smart Grids & Energy Systems
- Education / Intelligent Tutoring
- Finance & Trading

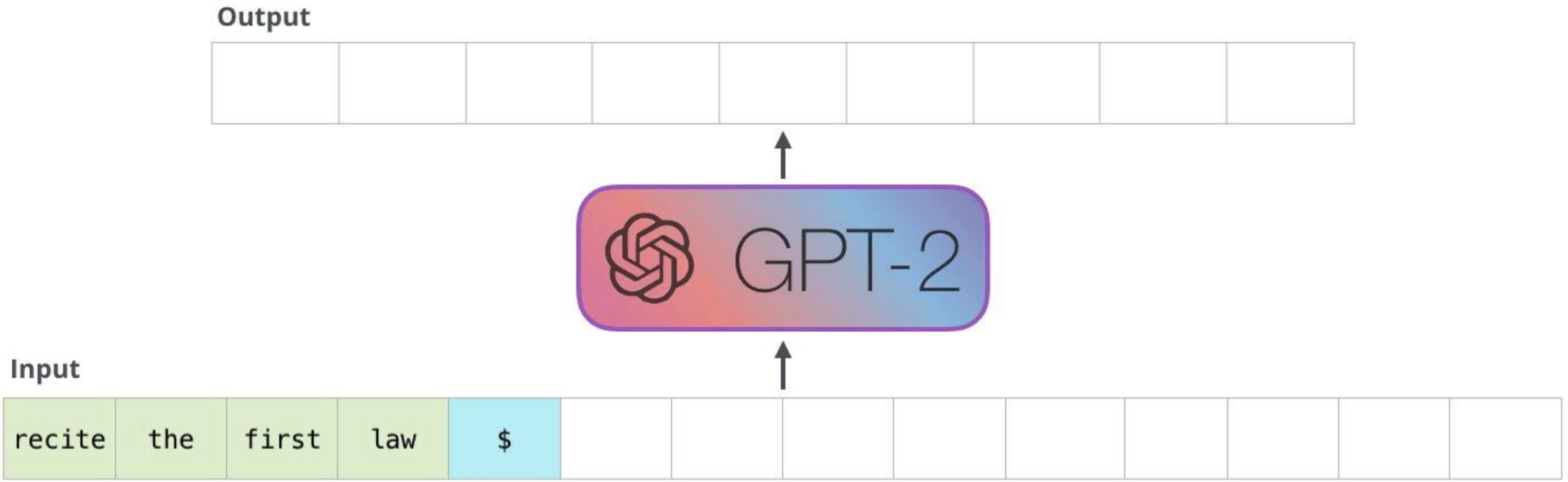




Decision Transformer



GPT-Style Transformers



The challenge of modeling sequences

A sequence may be arbitrarily long. We only have a finite number of parameters.

1. How do we model what comes next using only a finite number of parameters?
2. How do we build “expressive enough” models without cutting off the history at an arbitrary point (i.e. a Markov assumption)?
3. Can we build models that can encode items in a sequence in parallel?

The challenge of modeling sequences

A sequence may be arbitrarily long. We only have a finite number of parameters.

1. How do we model what comes next using only a finite number of parameters?

   ✓ Markov models ✓ Recurrent Neural Networks ✓ Transformers

2. How do we build “expressive enough” models without cutting off the history at an arbitrary point (i.e. a Markov assumption)?

   X Markov models ✓ Recurrent Neural Networks ✓ Transformers

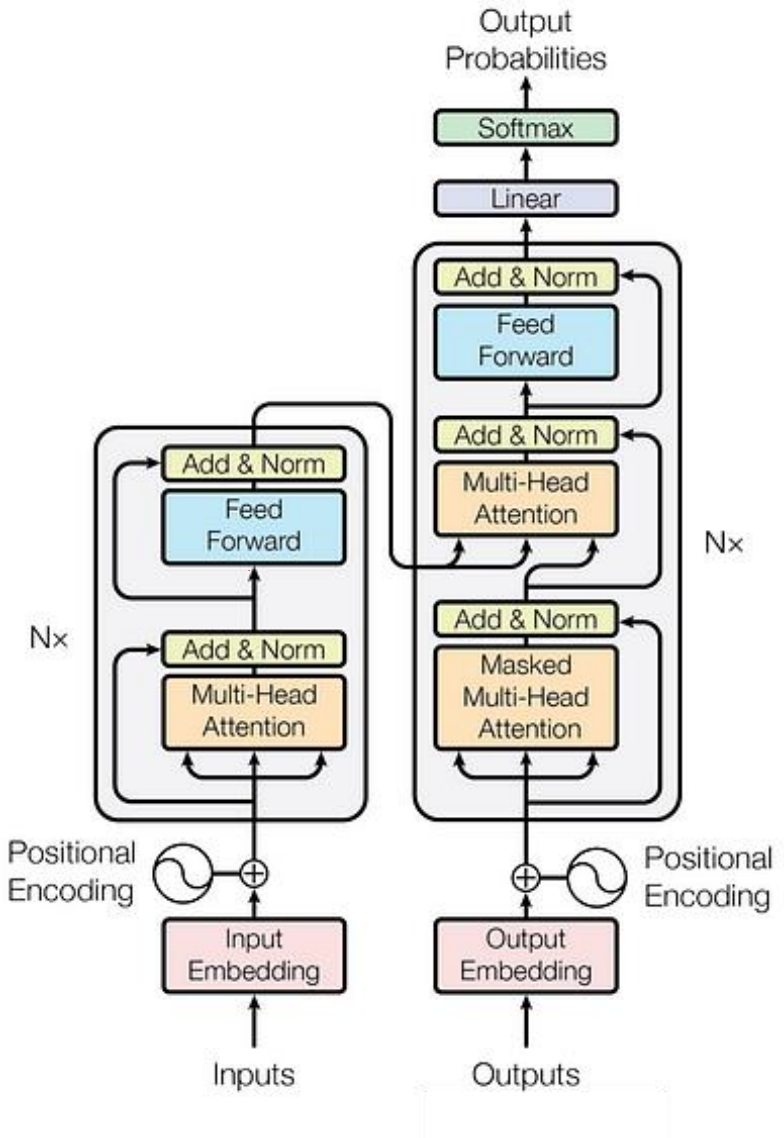
3. Can we build models that can encode items in a sequence in parallel?

   X Markov models X Recurrent Neural Networks ✓ Transformers

Transformer architecture: Vaswani et al 2017

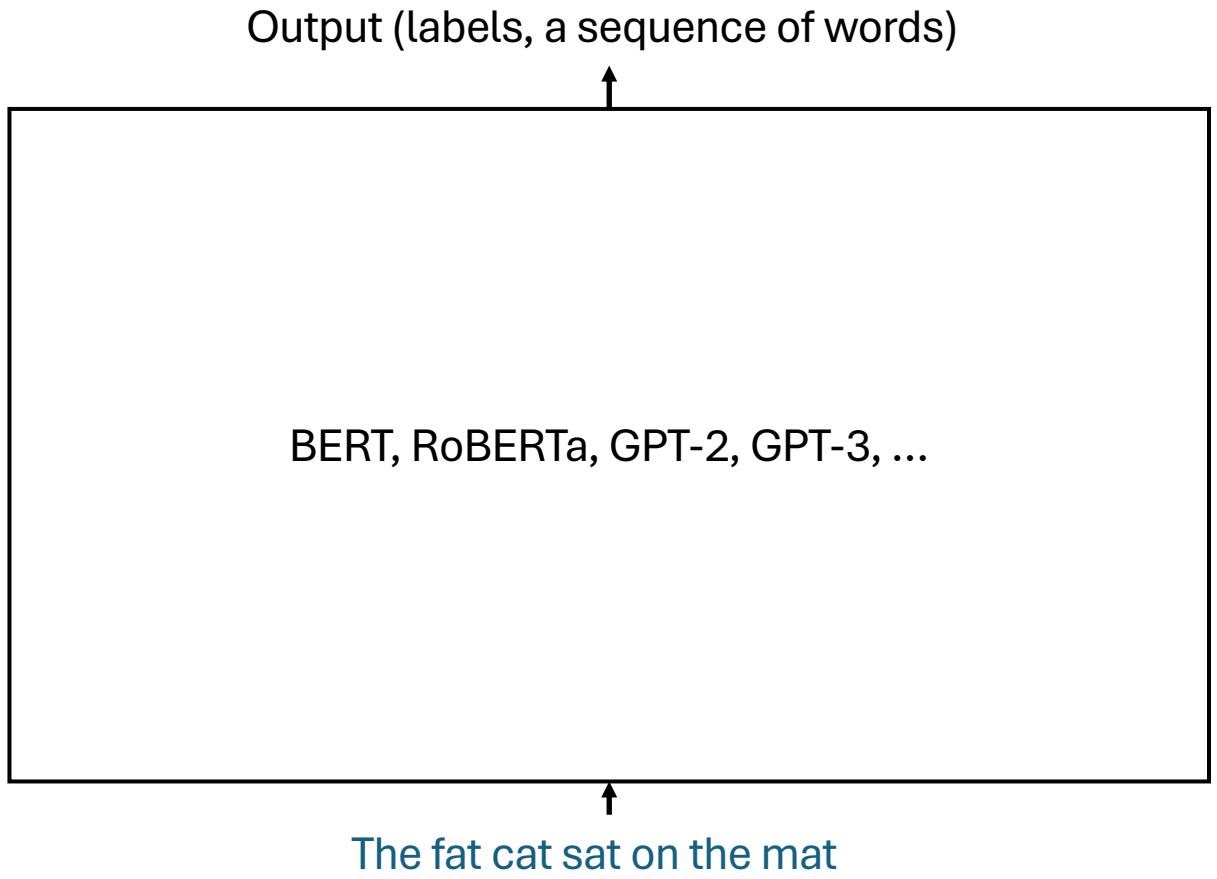
BERT

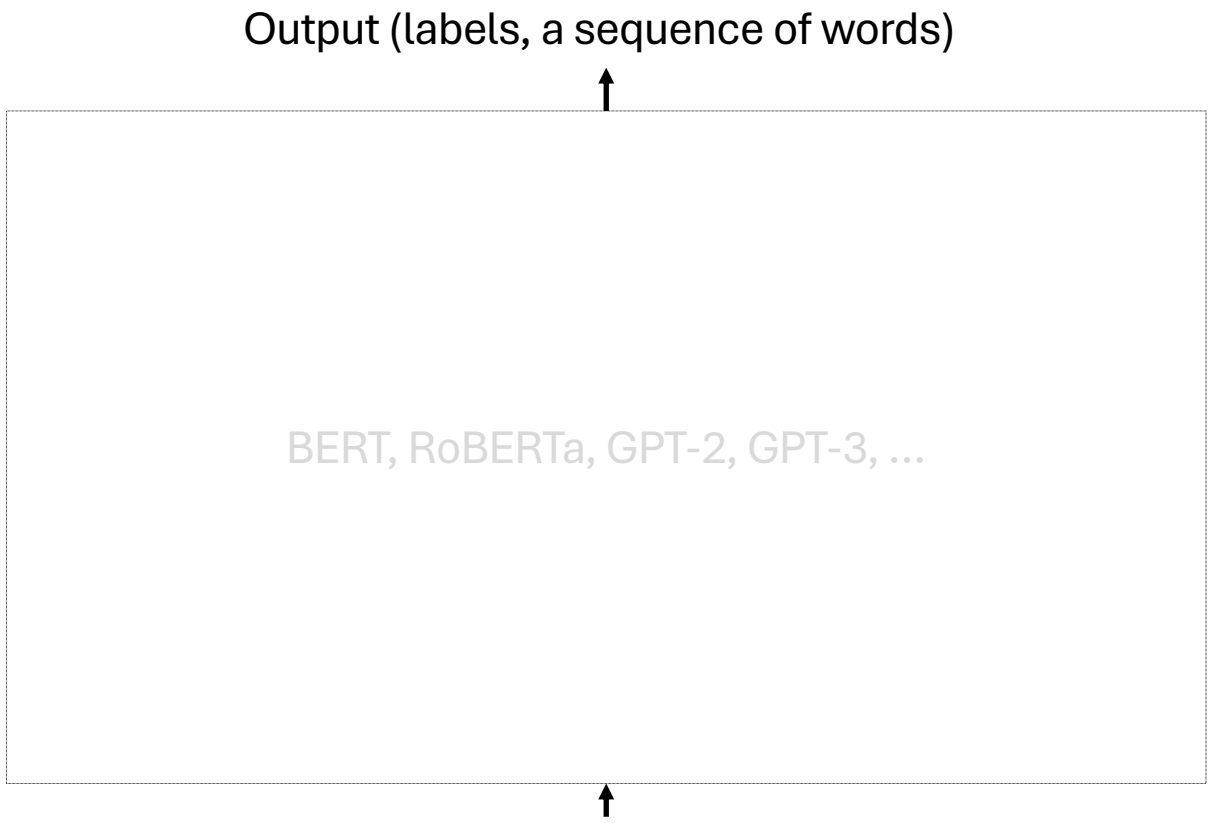
Encoder

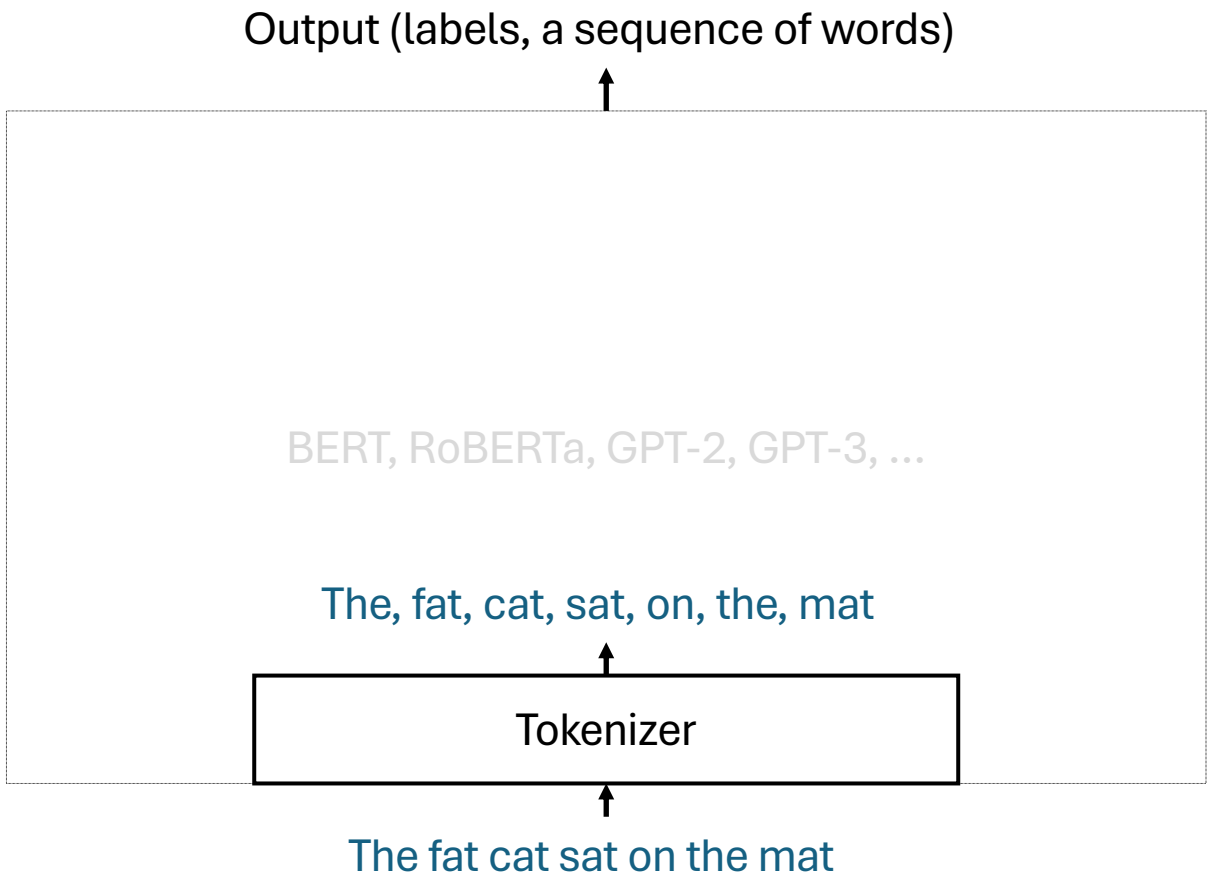


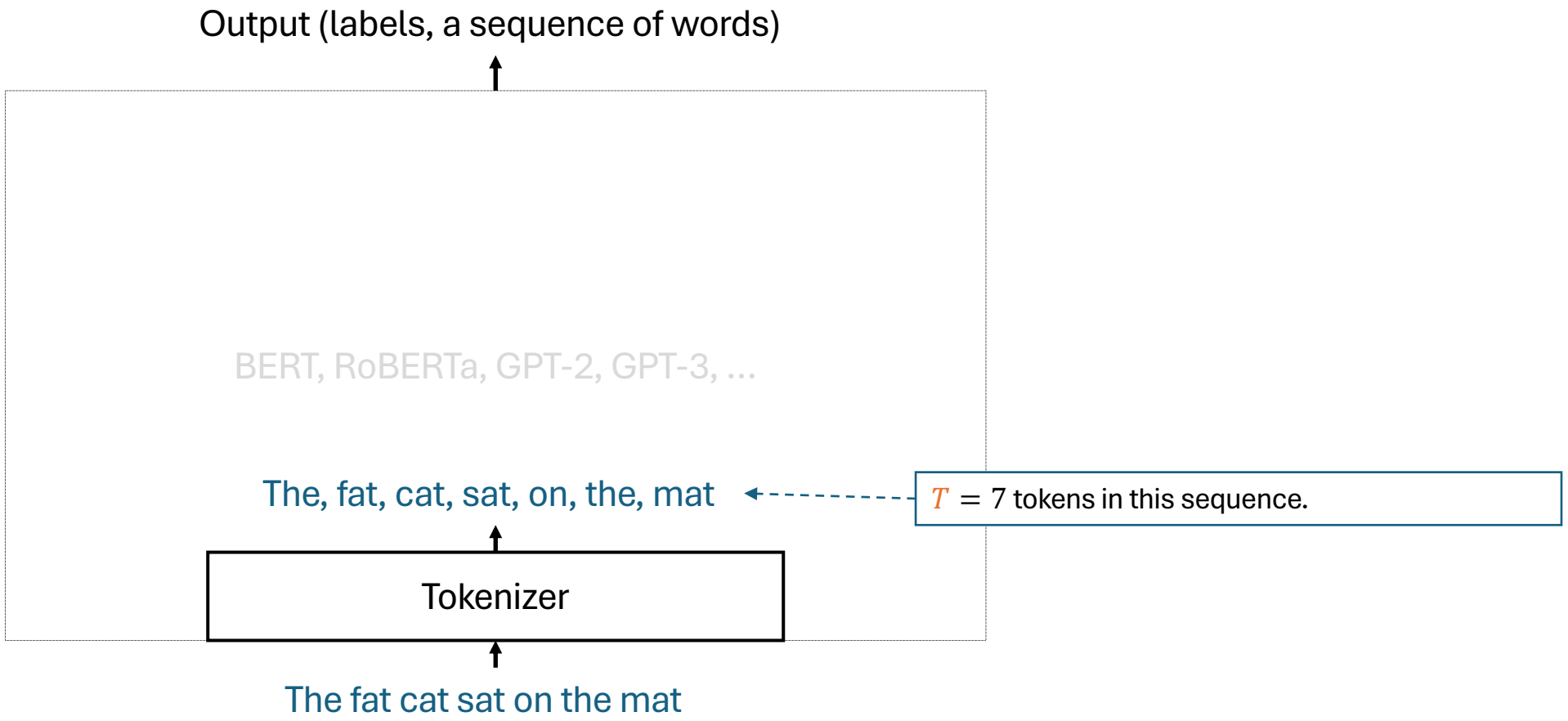
GPT

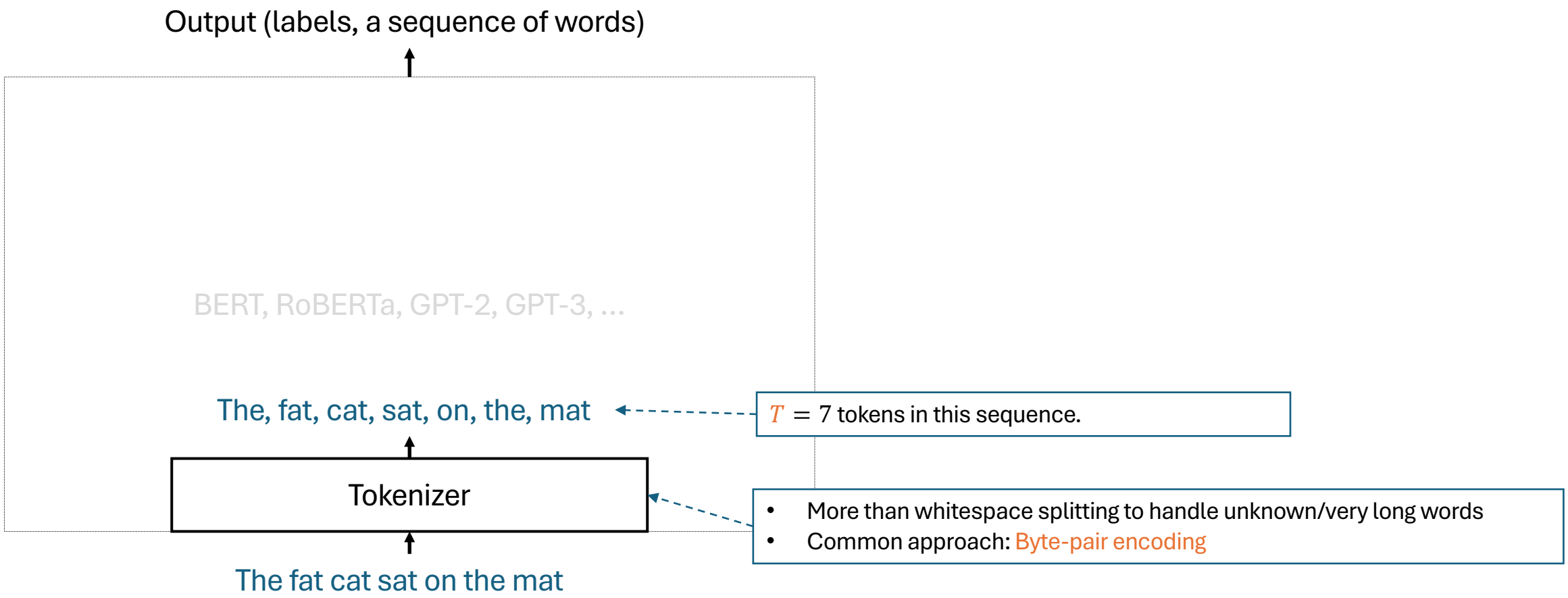
Decoder

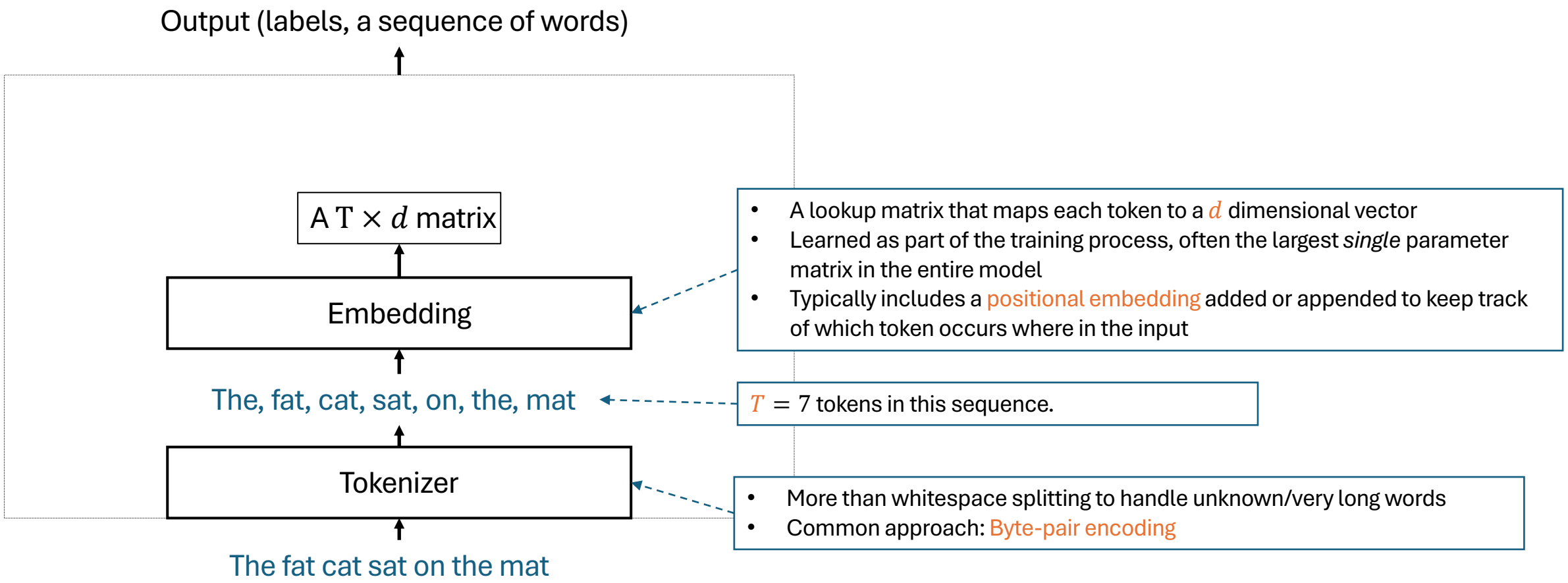

Output (labels, a sequence of words)
BERT, RoBERTa, GPT-2, GPT-3, ...
The fat cat sat on the mat

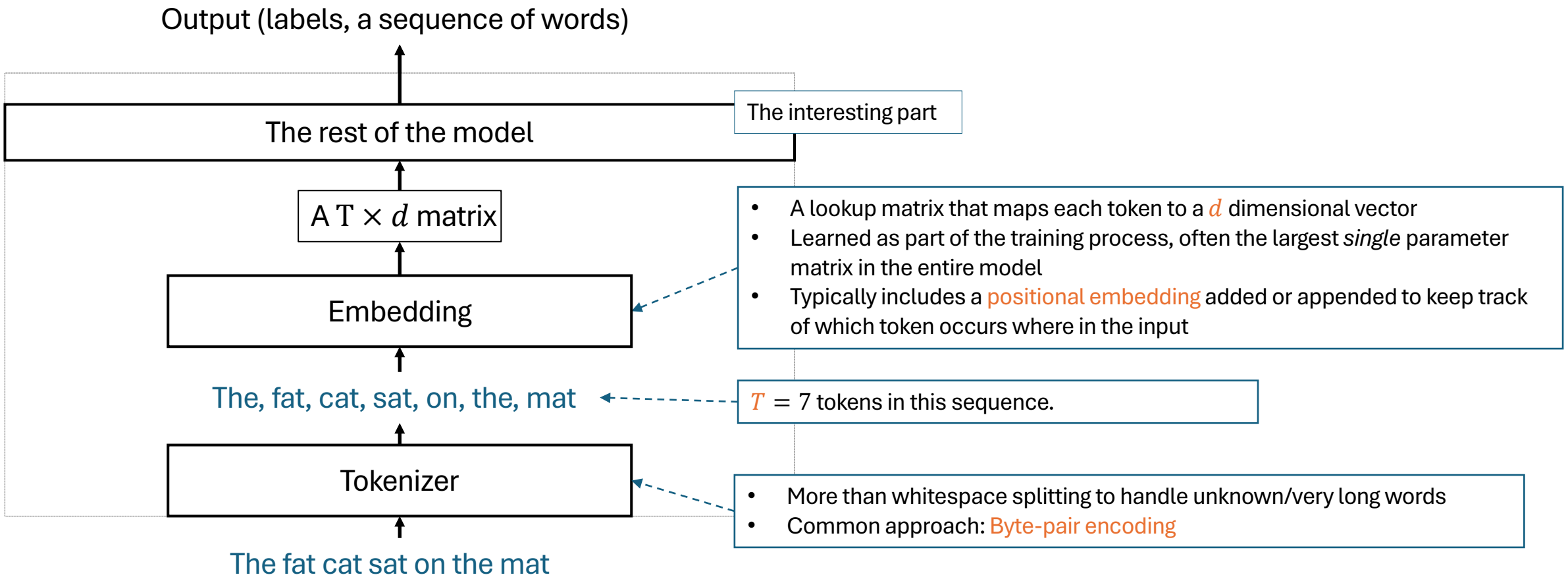

Output (labels, a sequence of words)
BERT, RoBERTa, GPT-2, GPT-3, ...
The fat cat sat on the mat

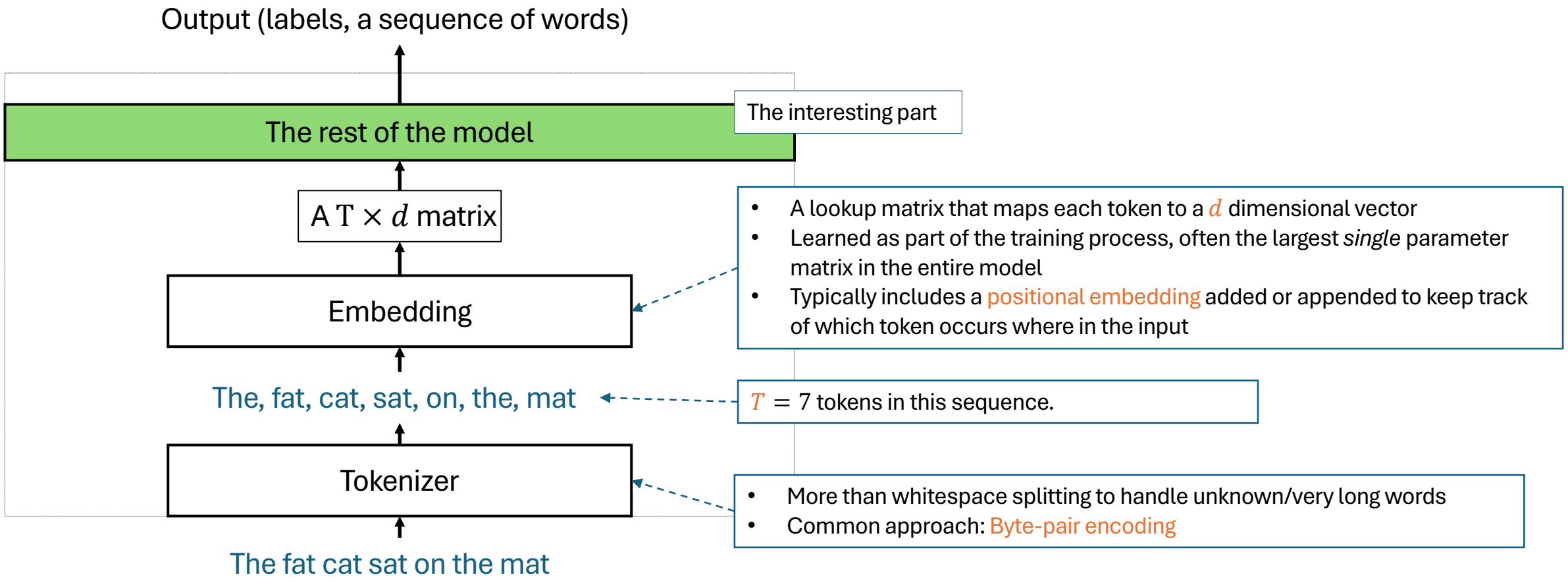

Output (labels, a sequence of words)
BERT, RoBERTa, GPT-2, GPT-3, ...
The, fat, cat, sat, on, the, mat
Tokenizer
The fat cat sat on the mat

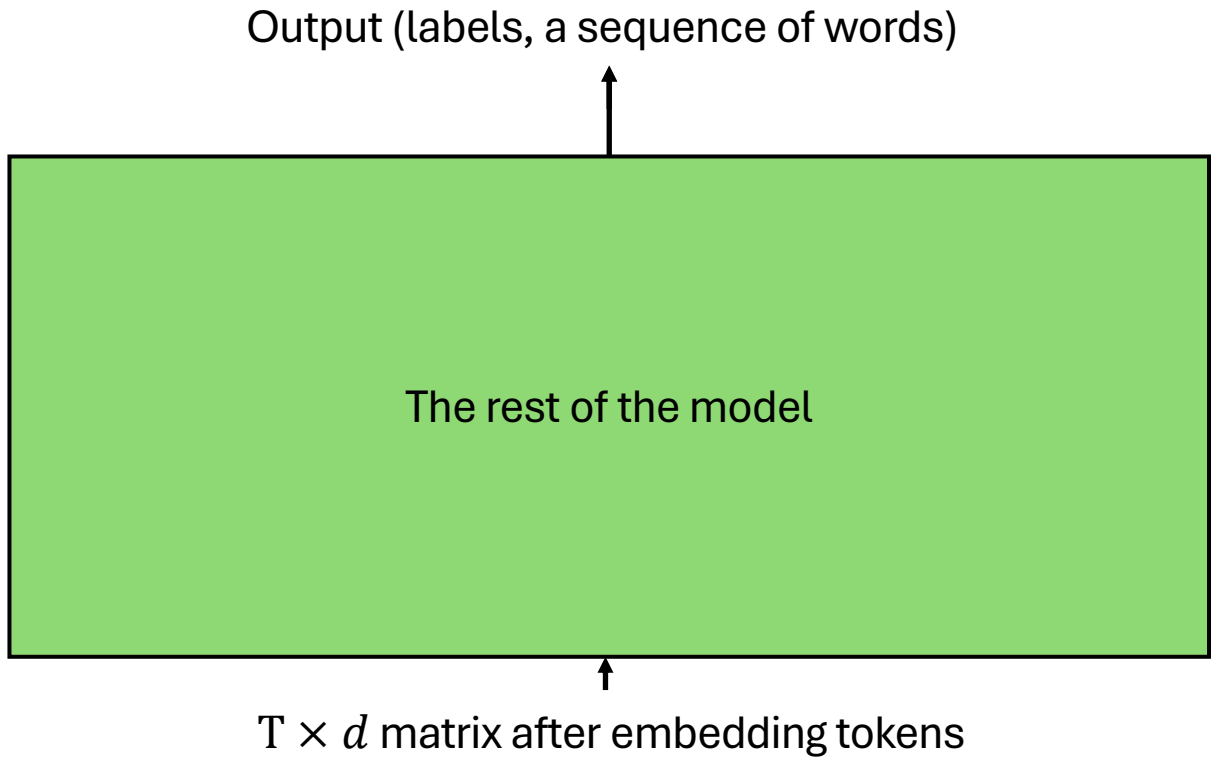

Output (labels, a sequence of words)
BERT, RoBERTa, GPT-2, GPT-3, ...
The, fat, cat, sat, on, the, mat
T = 7 tokens in this sequence.
Tokenizer
The fat cat sat on the mat

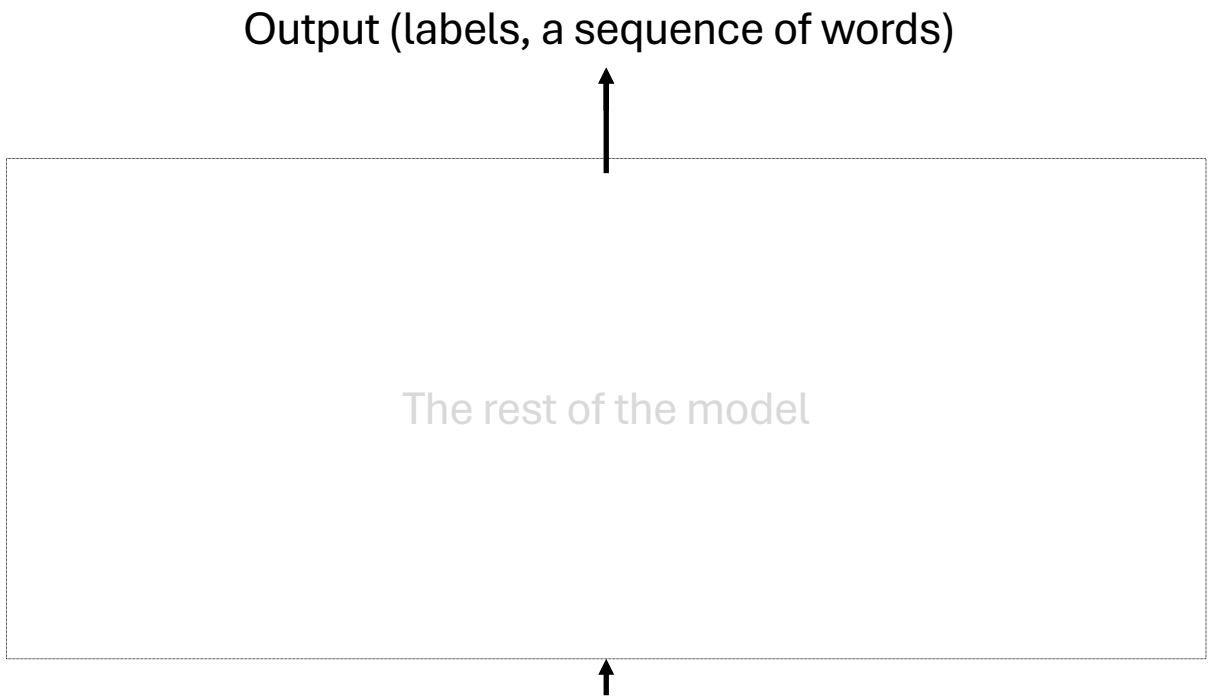

Output (labels, a sequence of words)
BERT, RoBERTa, GPT-2, GPT-3, …
The, fat, cat, sat, on, the, mat
T = 7 tokens in this sequence.
Tokenizer
More than whitespace splitting to handle unknown/very long words
Common approach: Byte-pair encoding
The fat cat sat on the mat


Output (labels, a sequence of words)
A T × d matrix
Embedding
The, fat, cat, sat, on, the, mat
Tokenizer
The fat cat sat on the mat
• A lookup matrix that maps each token to a d dimensional vector
• Learned as part of the training process, often the largest single parameter matrix in the entire model
• Typically includes a positional embedding added or appended to keep track of which token occurs where in the input
T = 7 tokens in this sequence.
• More than whitespace splitting to handle unknown/very long words
• Common approach: Byte-pair encoding

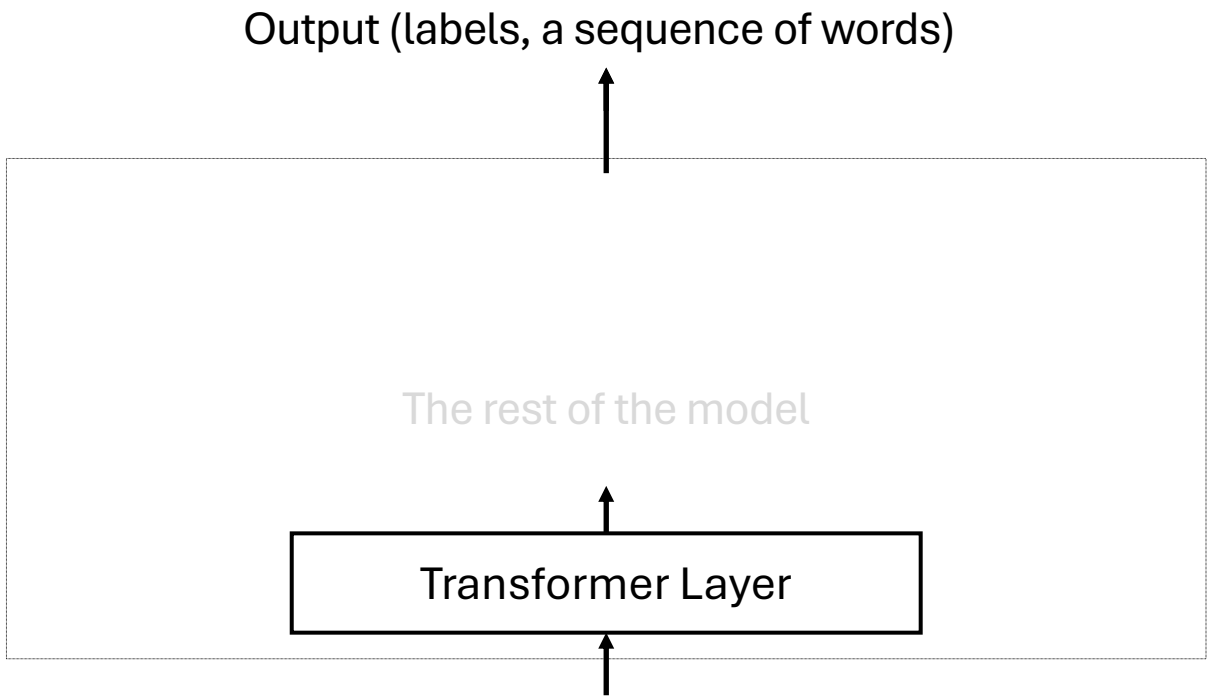

Output (labels, a sequence of words)
The rest of the model
The interesting part
A T × d matrix
Embedding
• A lookup matrix that maps each token to a d dimensional vector
• Learned as part of the training process, often the largest single parameter matrix in the entire model
• Typically includes a positional embedding added or appended to keep track of which token occurs where in the input
The, fat, cat, sat, on, the, mat
T = 7 tokens in this sequence.
Tokenizer
• More than whitespace splitting to handle unknown/very long words
• Common approach: Byte-pair encoding
The fat cat sat on the mat


Output (labels, a sequence of words)
The rest of the model
The interesting part
A T × d matrix
Embedding
• A lookup matrix that maps each token to a d dimensional vector
• Learned as part of the training process, often the largest single parameter matrix in the entire model
• Typically includes a positional embedding added or appended to keep track of which token occurs where in the input
The, fat, cat, sat, on, the, mat
T = 7 tokens in this sequence.
Tokenizer
• More than whitespace splitting to handle unknown/very long words
• Common approach: Byte-pair encoding
The fat cat sat on the mat

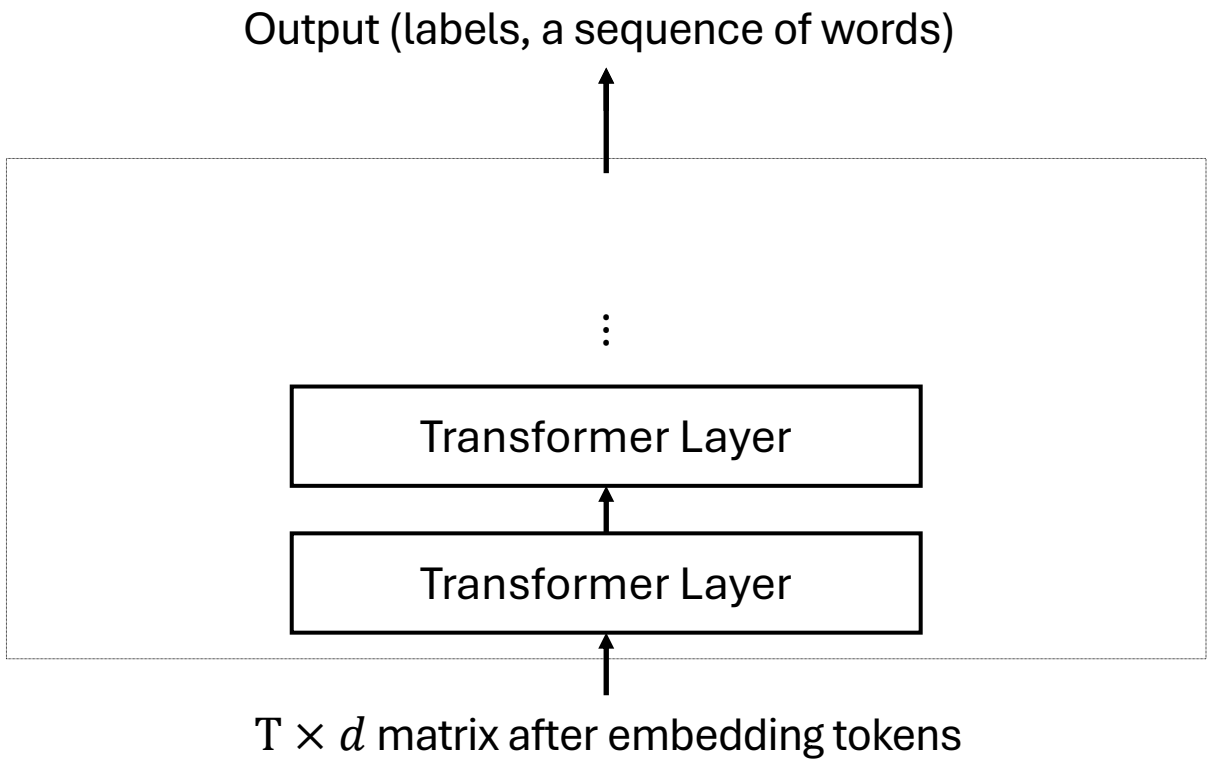

Output (labels, a sequence of words)
The rest of the model
T × d matrix after embedding tokens



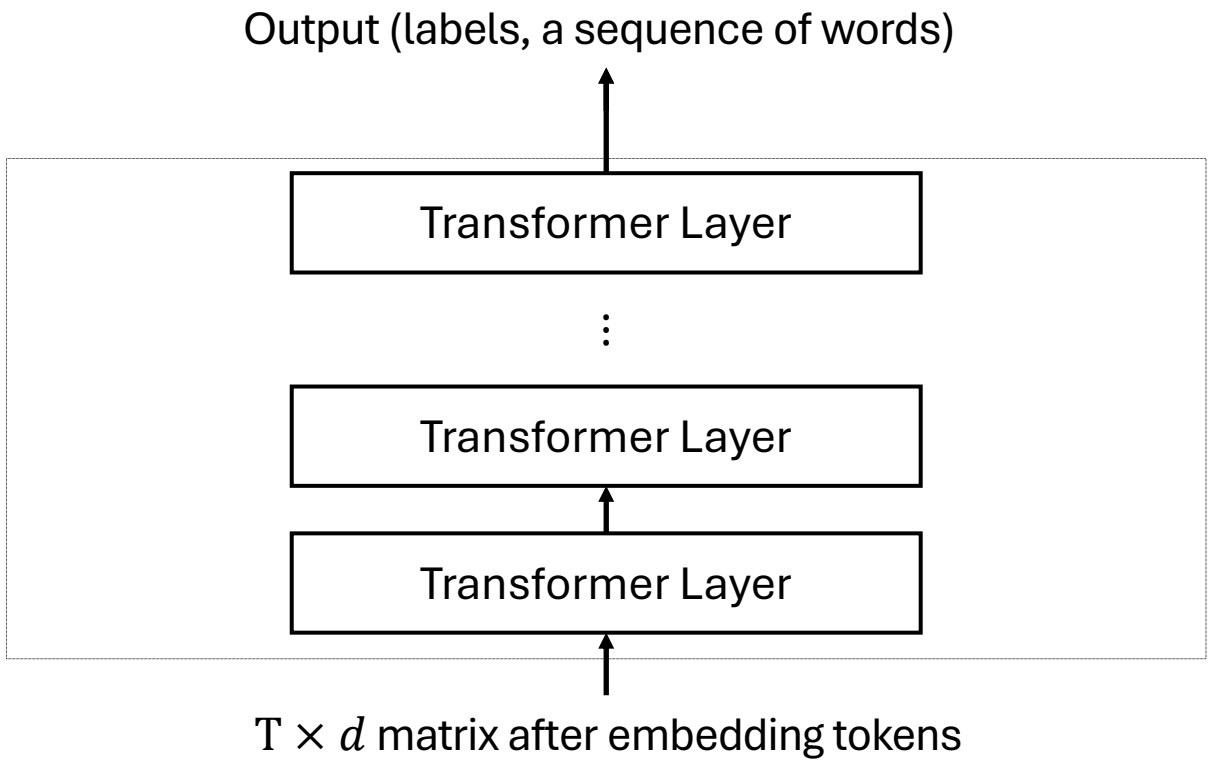
T: Sequence length
d: Embedding size



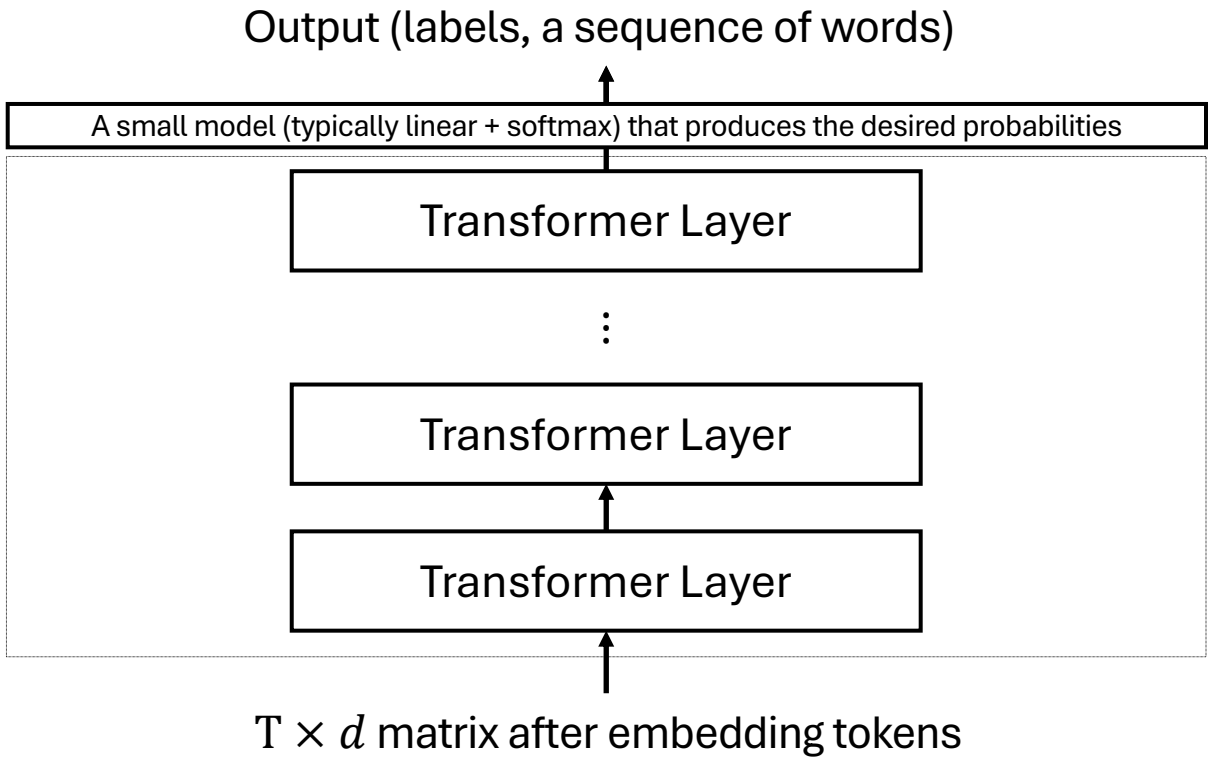
T: Sequence length
d: Embedding size



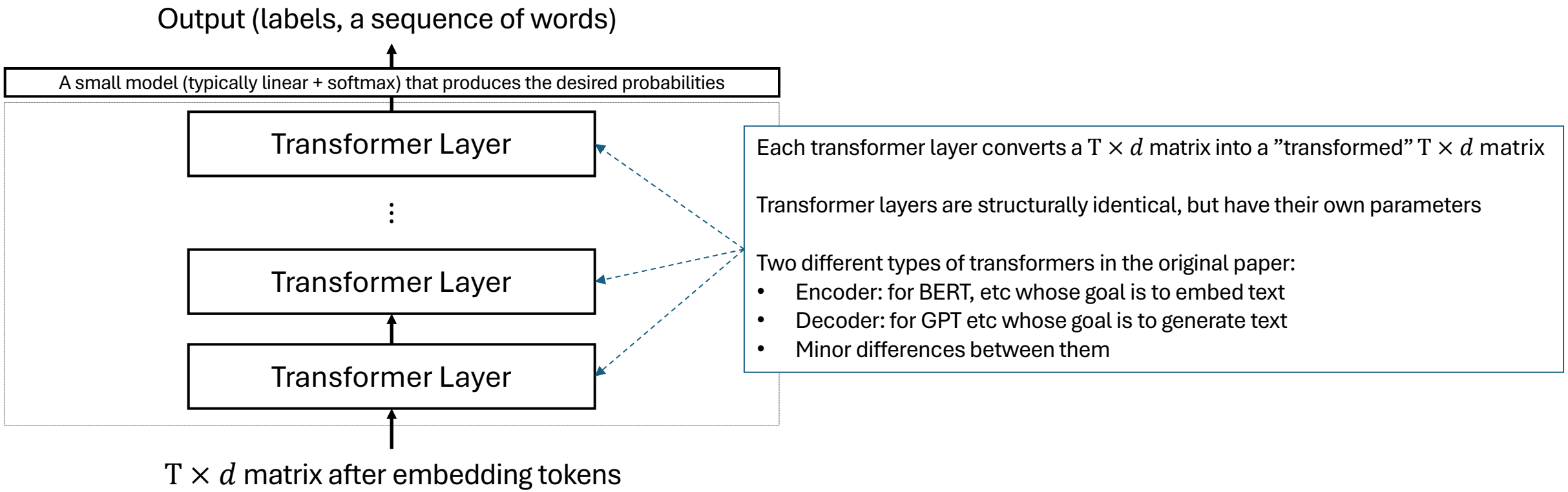
T: Sequence length
d: Embedding size



T: Sequence length
d: Embedding size

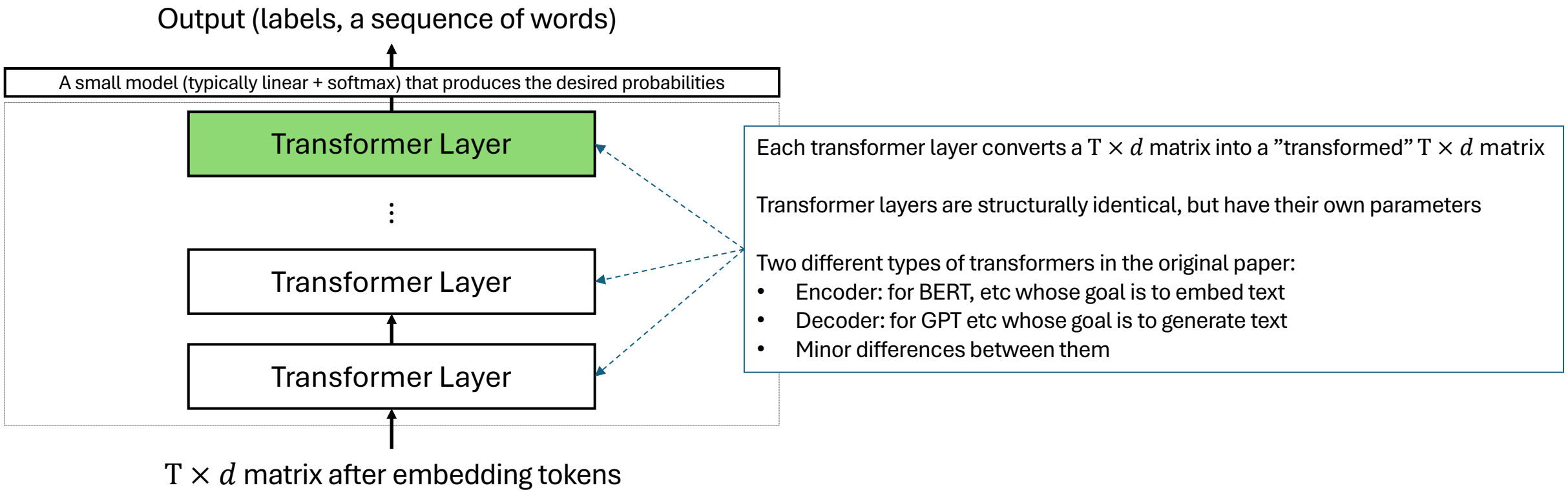


T: Sequence length
d: Embedding size



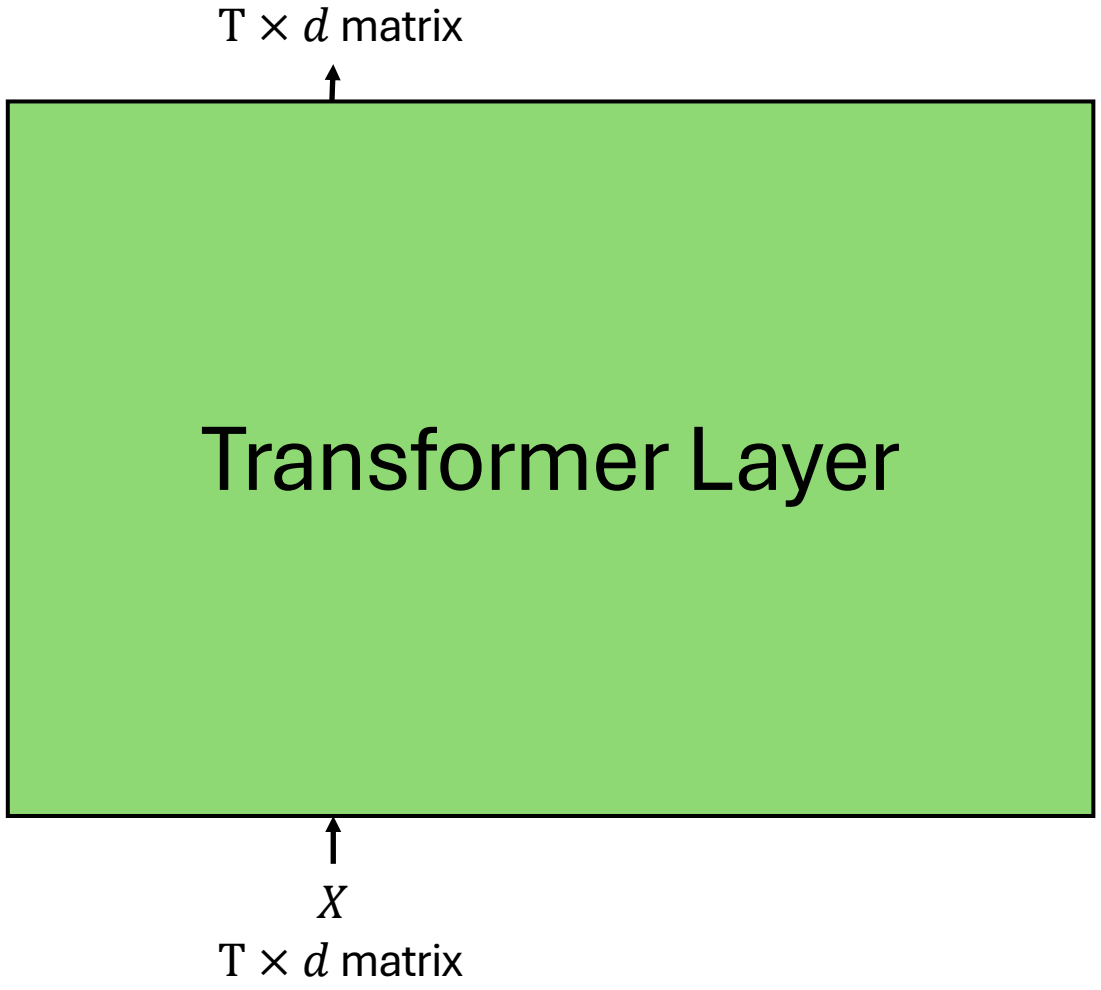
T: Sequence length

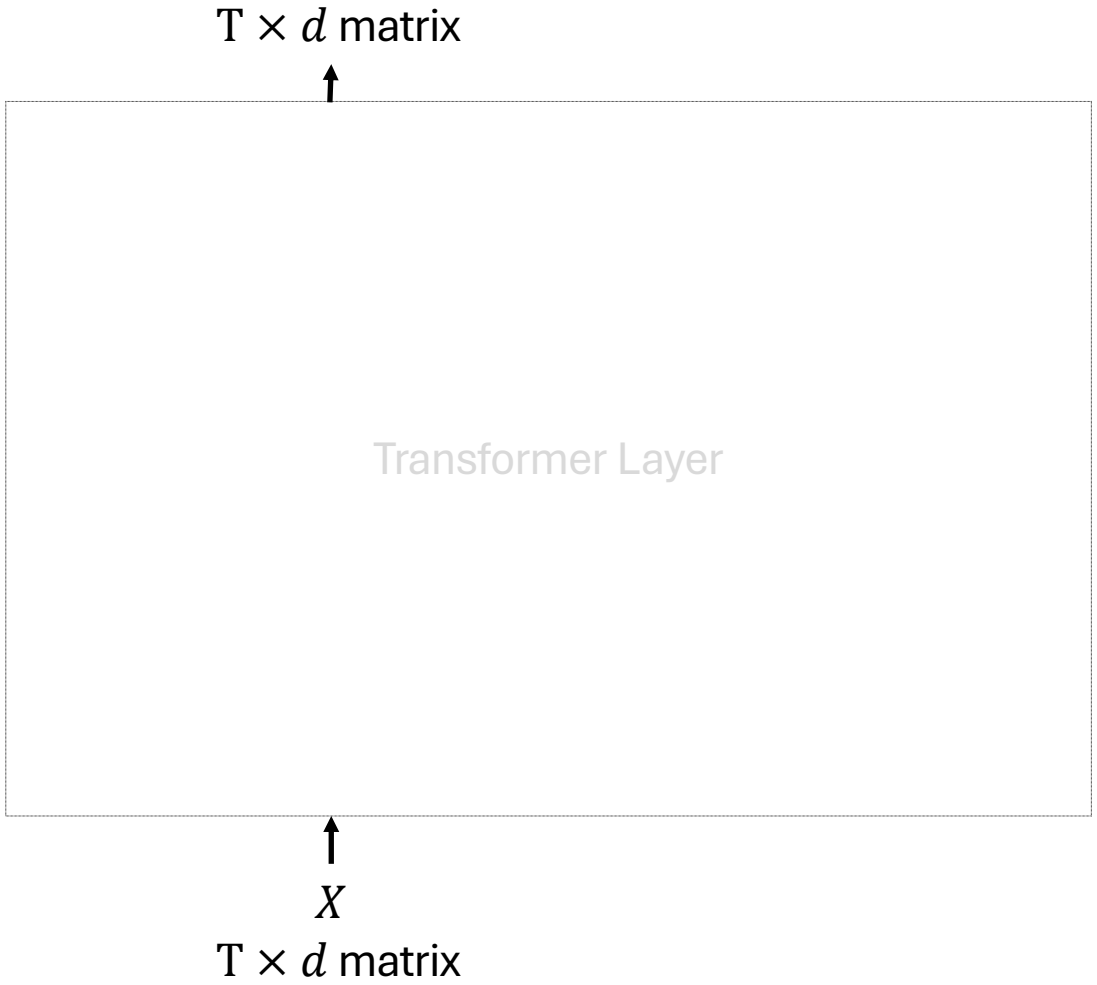
d: Embedding size

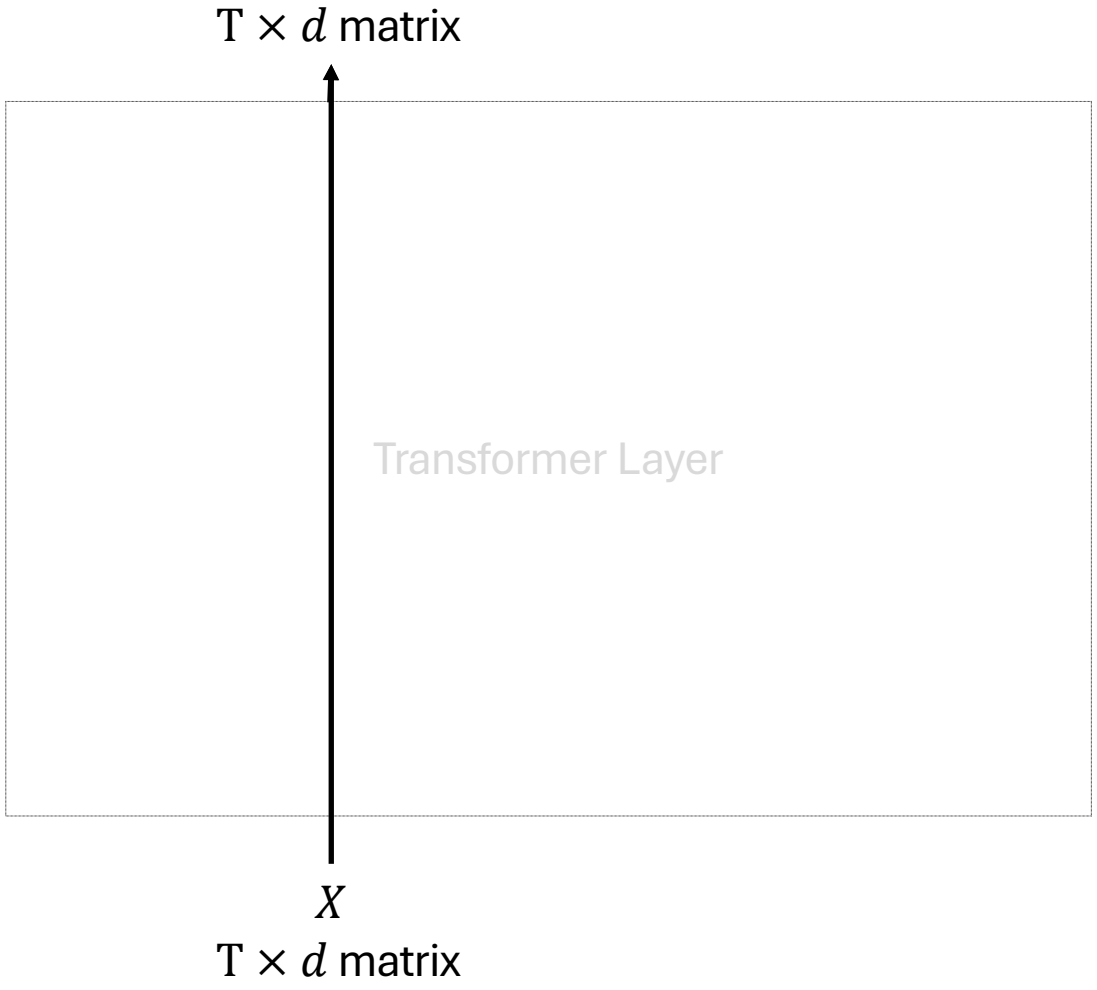


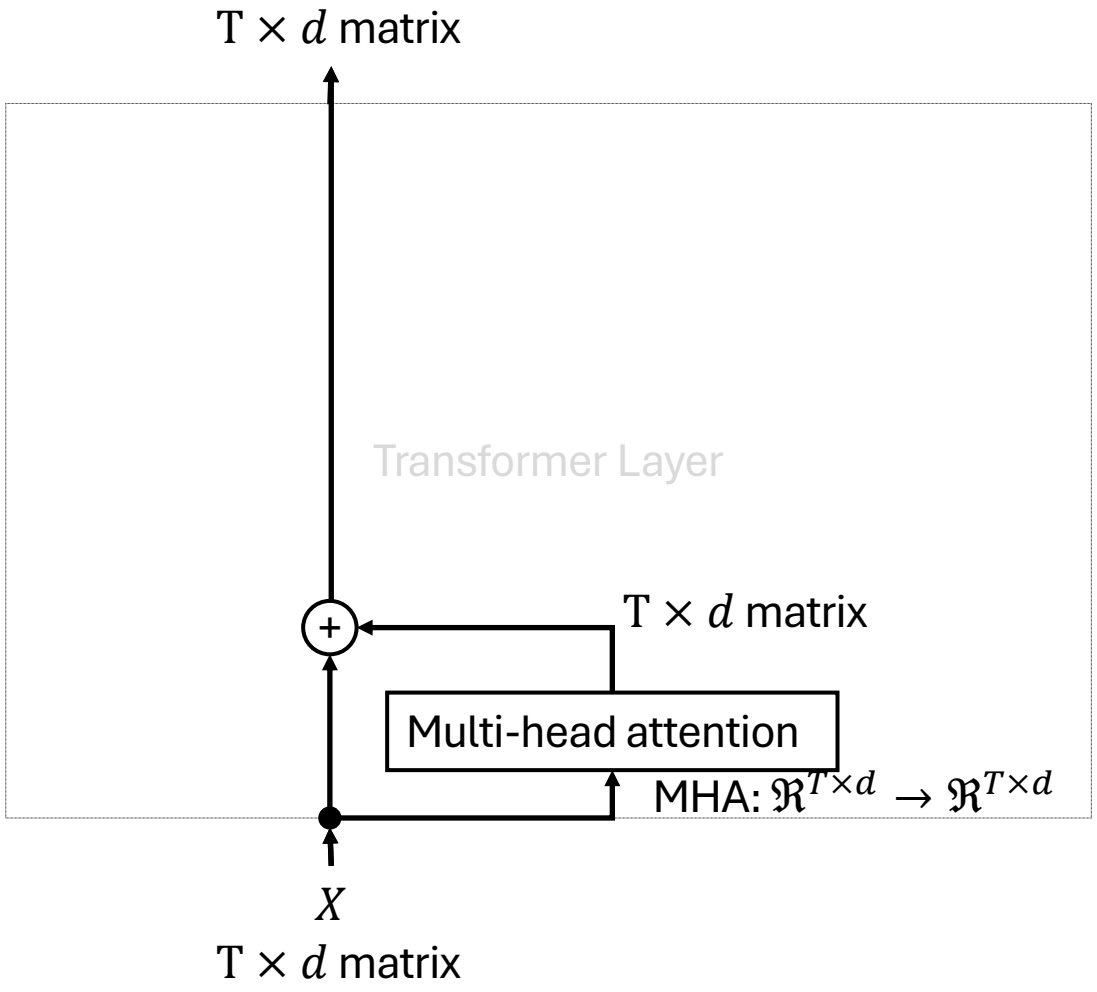
T: Sequence length

d: Embedding size

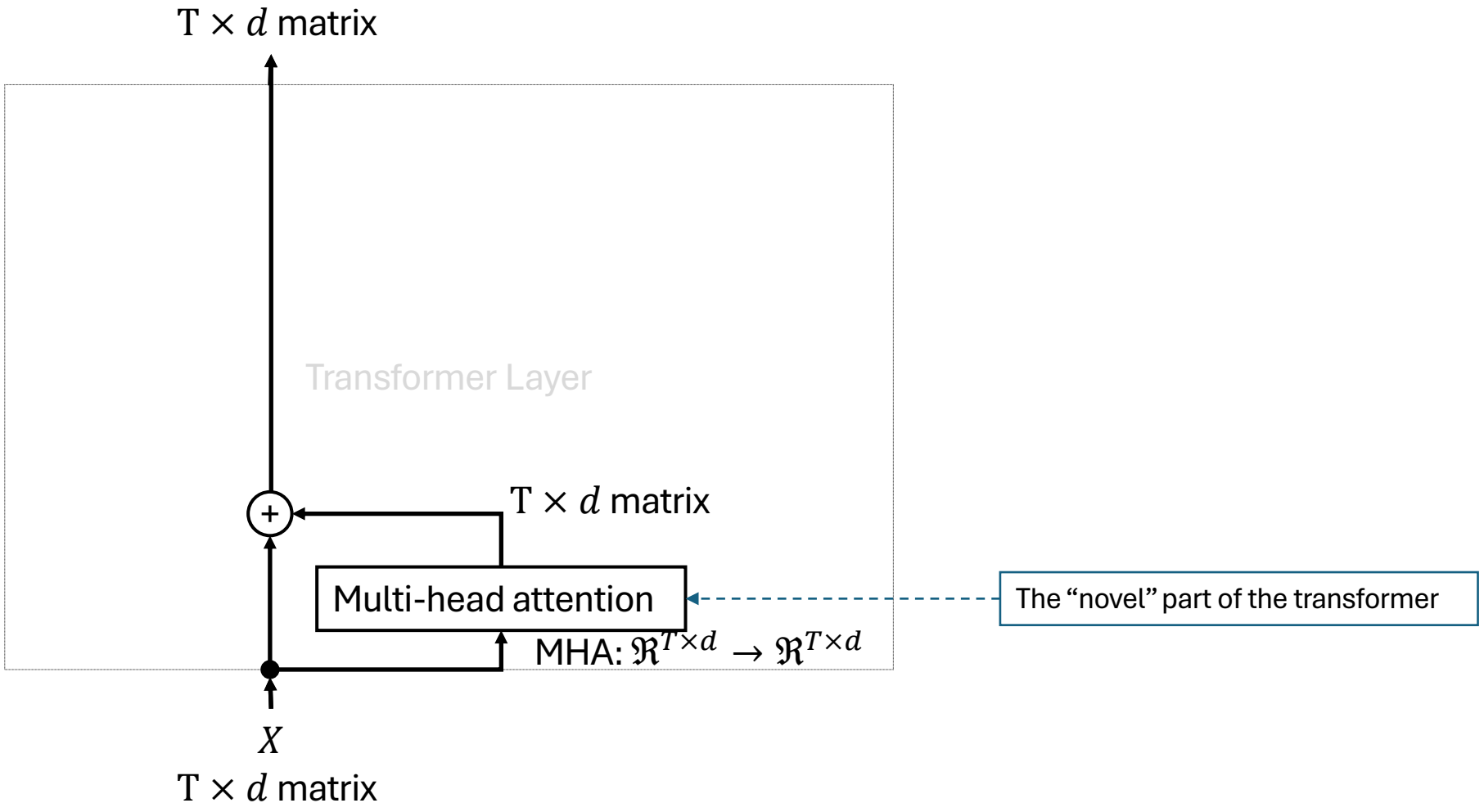

T × d matrix
Transformer Layer
X
T × d matrix

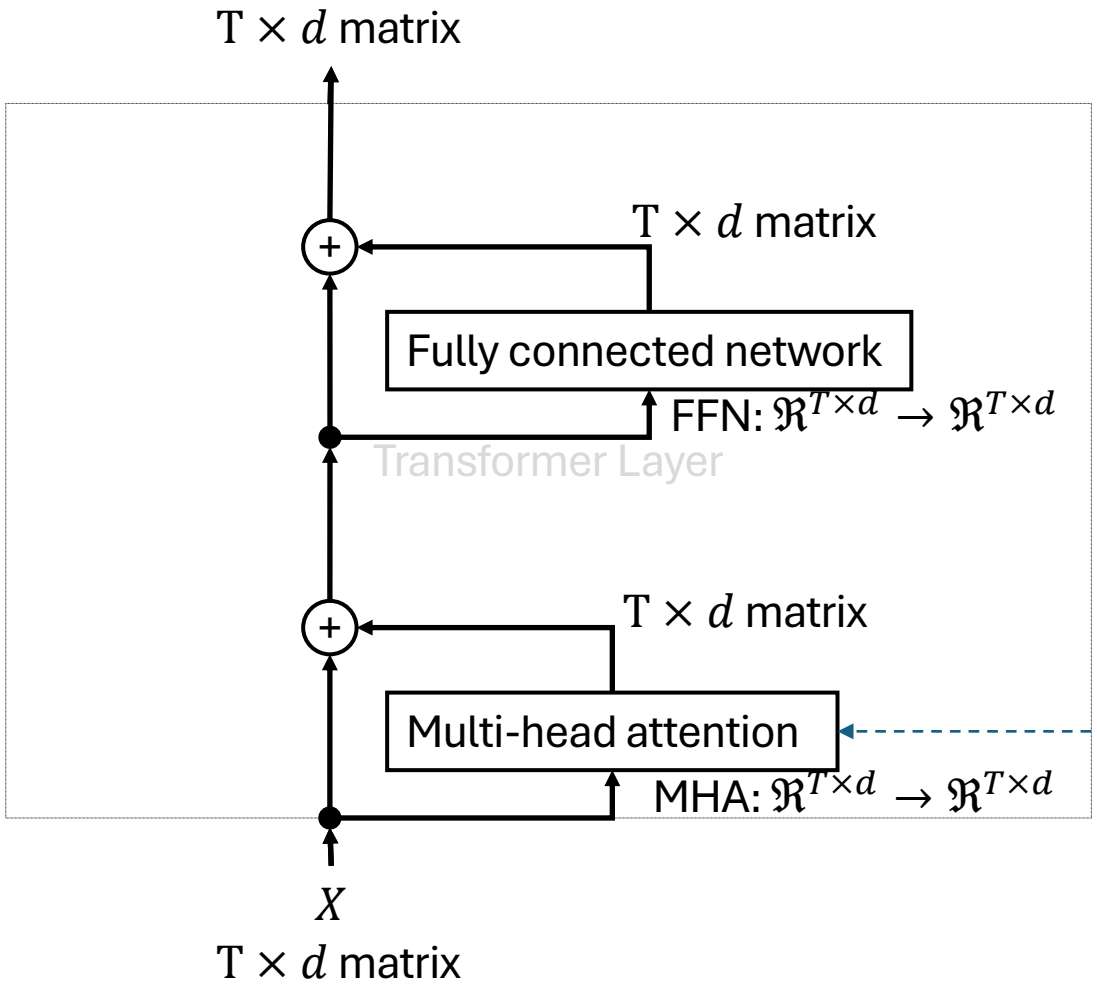

T × d matrix
Transformer Layer
X
T × d matrix


T × d matrix
Transformer Layer
X
T × d matrix



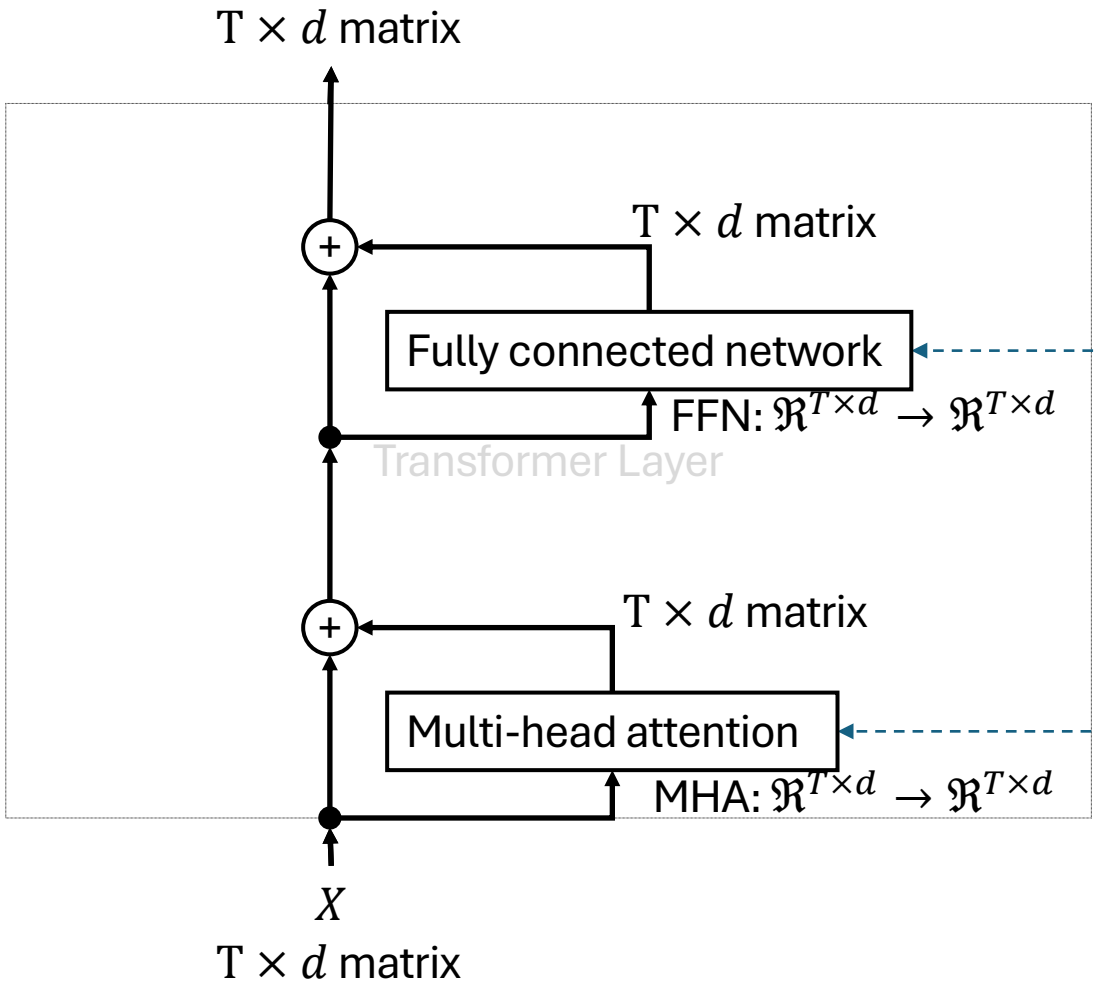
A residual connection


T × d matrix
Transformer Layer
T × d matrix
Multi-head attention
MHA: $\Re^{T\times d} \rightarrow \Re^{T\times d}$
X
T × d matrix
The “novel” part of the transformer



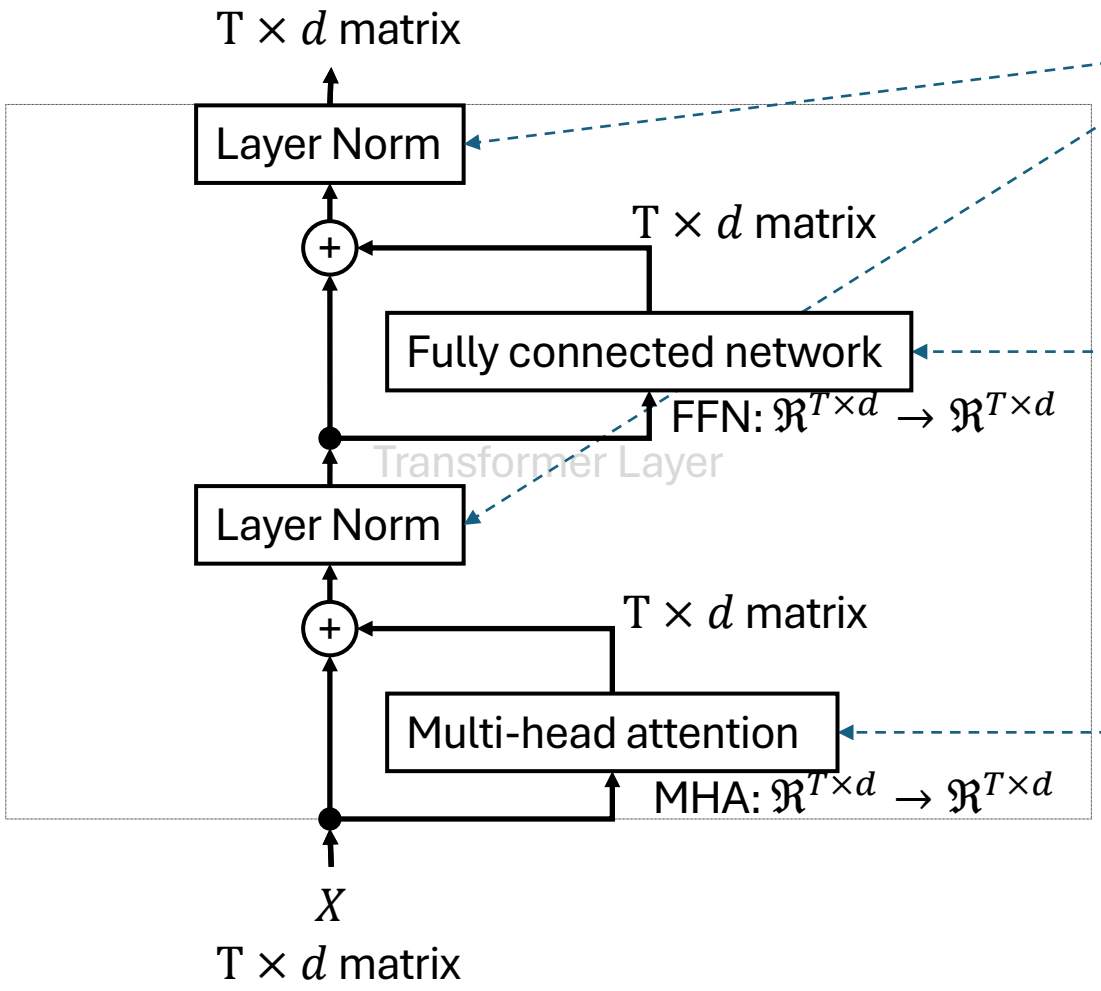
A residual connection

The "novel" part of the transformer



- A standard two layer neural network with GELU, ReLU (or sometimes other) activation, applied to each row separately
  - Trivially parallelizable over the rows
- Hidden layer can have dimensionality larger than d (e.g. $4d$)

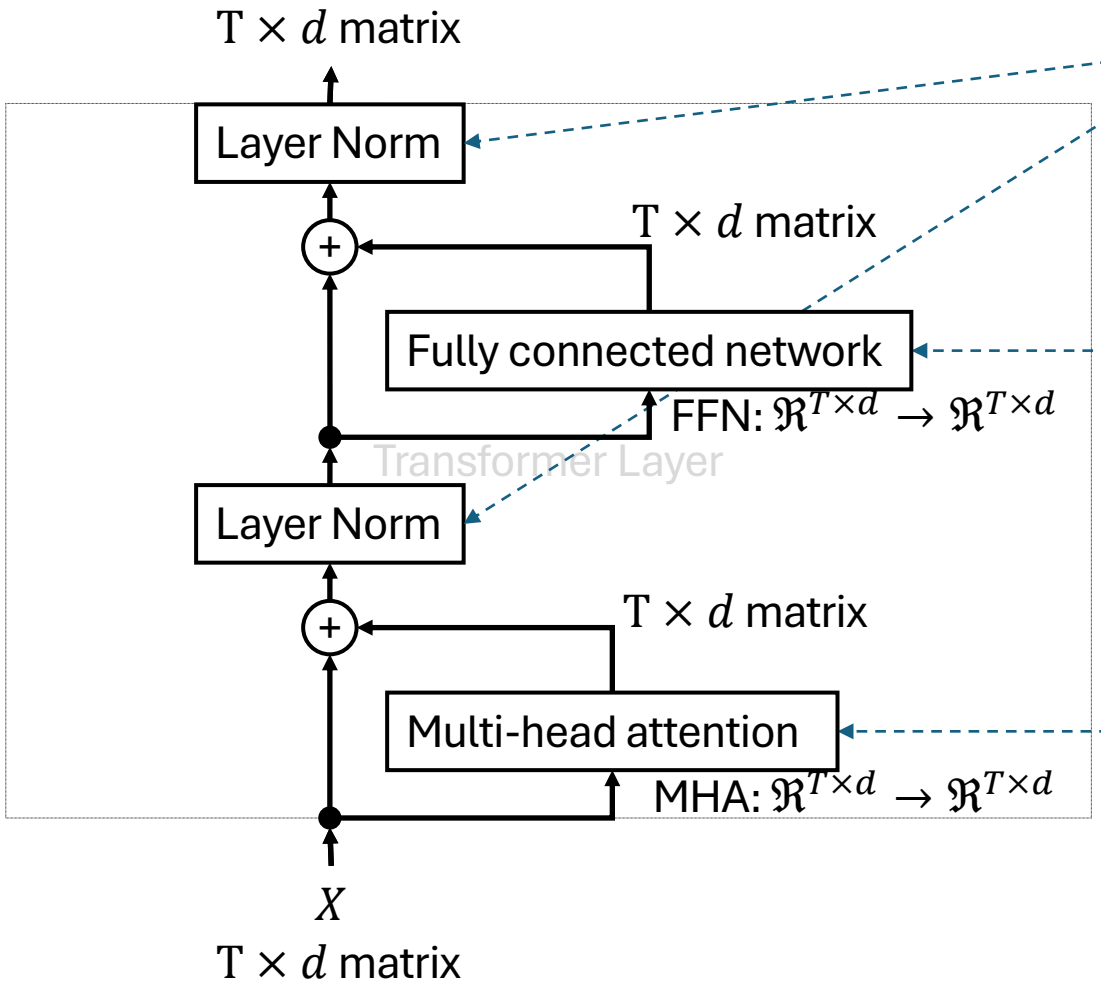
The "novel" part of the transformer



- Normalizes each column of the input matrix so that the elements have zero mean and unit variance
- Leads to more stable optimization

- A standard two layer neural network with GELU, ReLU (or sometimes other) activation, applied to each row separately
  - Trivially parallelizable over the rows
- Hidden layer can have dimensionality larger than d (e.g. $4d$)

The "novel" part of the transformer

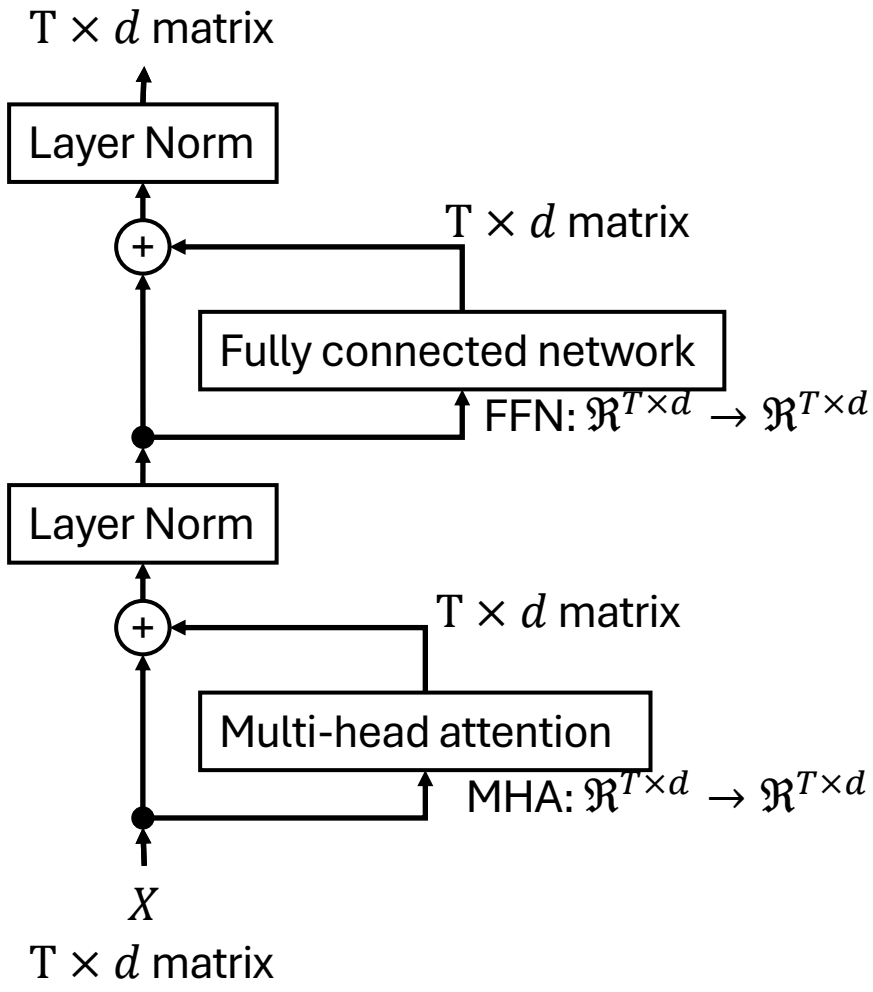


- Normalizes each column of the input matrix so that the elements have zero mean and unit variance
- Leads to more stable optimization

- A standard two layer neural network with GELU, ReLU (or sometimes other) activation, applied to each row separately
  - Trivially parallelizable over the rows
- Hidden layer can have dimensionality larger than d (e.g. $4d$)

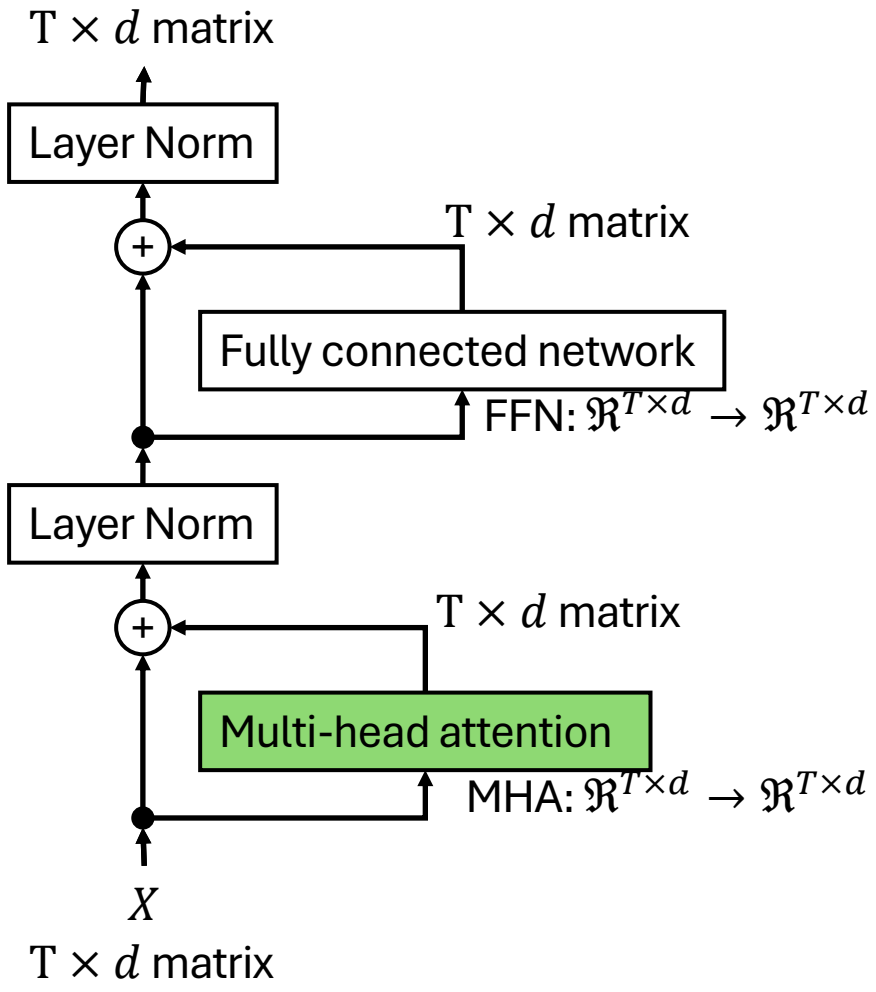
The "novel" part of the transformer

This is the transformer encoder. The decoder transformer has a bit more detail. We will encounter the details later



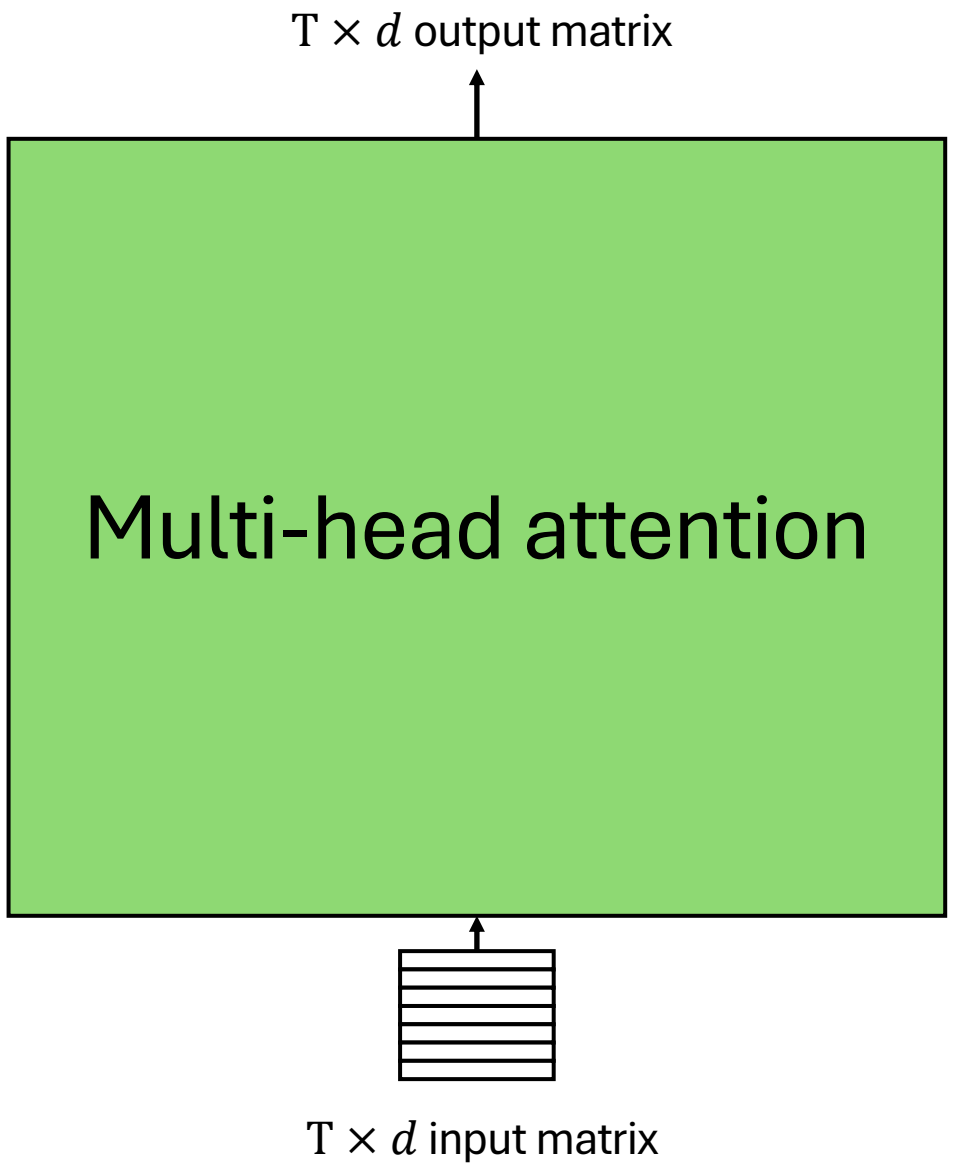

```python
def transformer_layer(X):
    X1 = layer_norm1(X + multi_head_attention(X))
    X2 = layer_norm2(X1 + fully_connected(X1))
    return X2
```

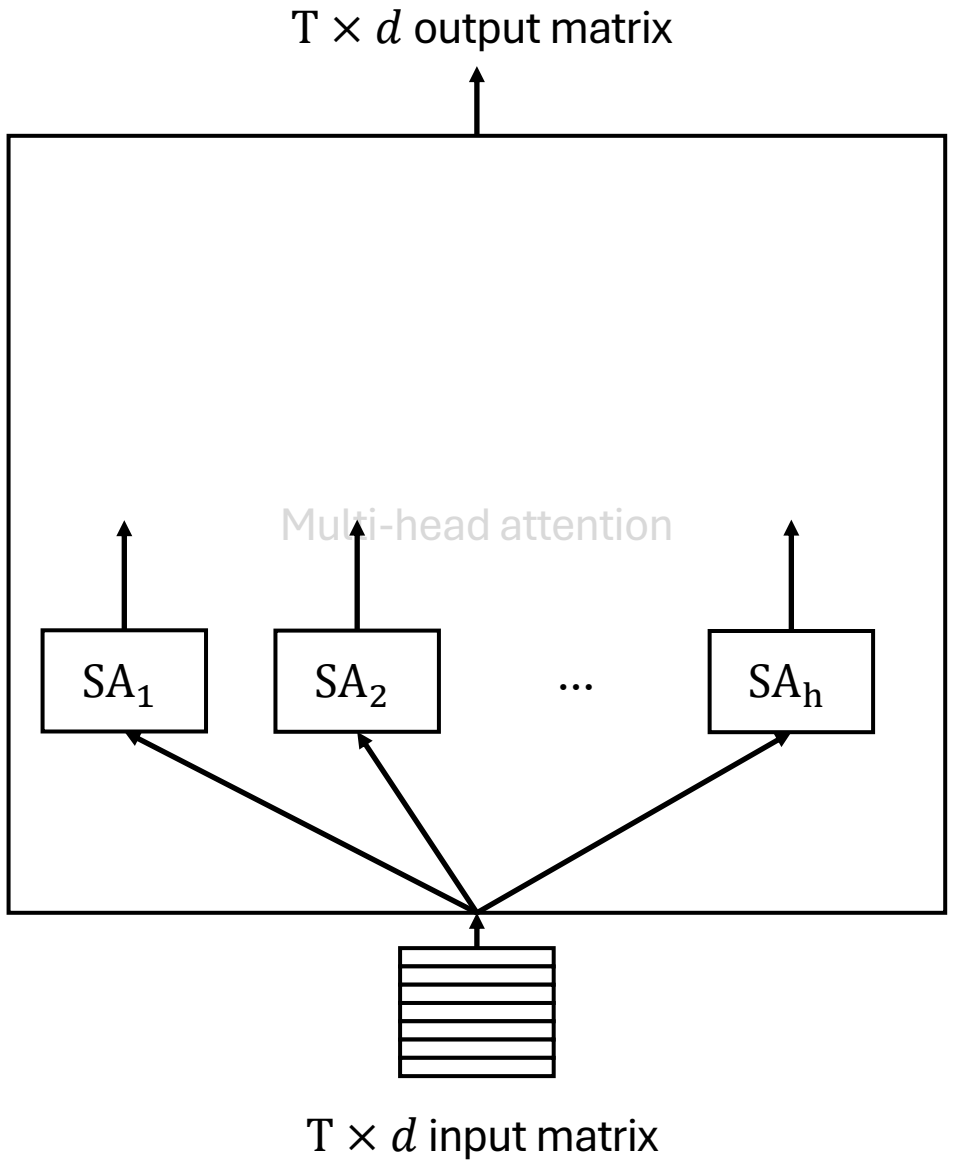
$$X_1 = \text{LayerNorm}\big(X + MHA(X)\big)$$
$$\text{Result} = \text{LayerNorm}\big(X_1 + FFN(X_1)\big)$$



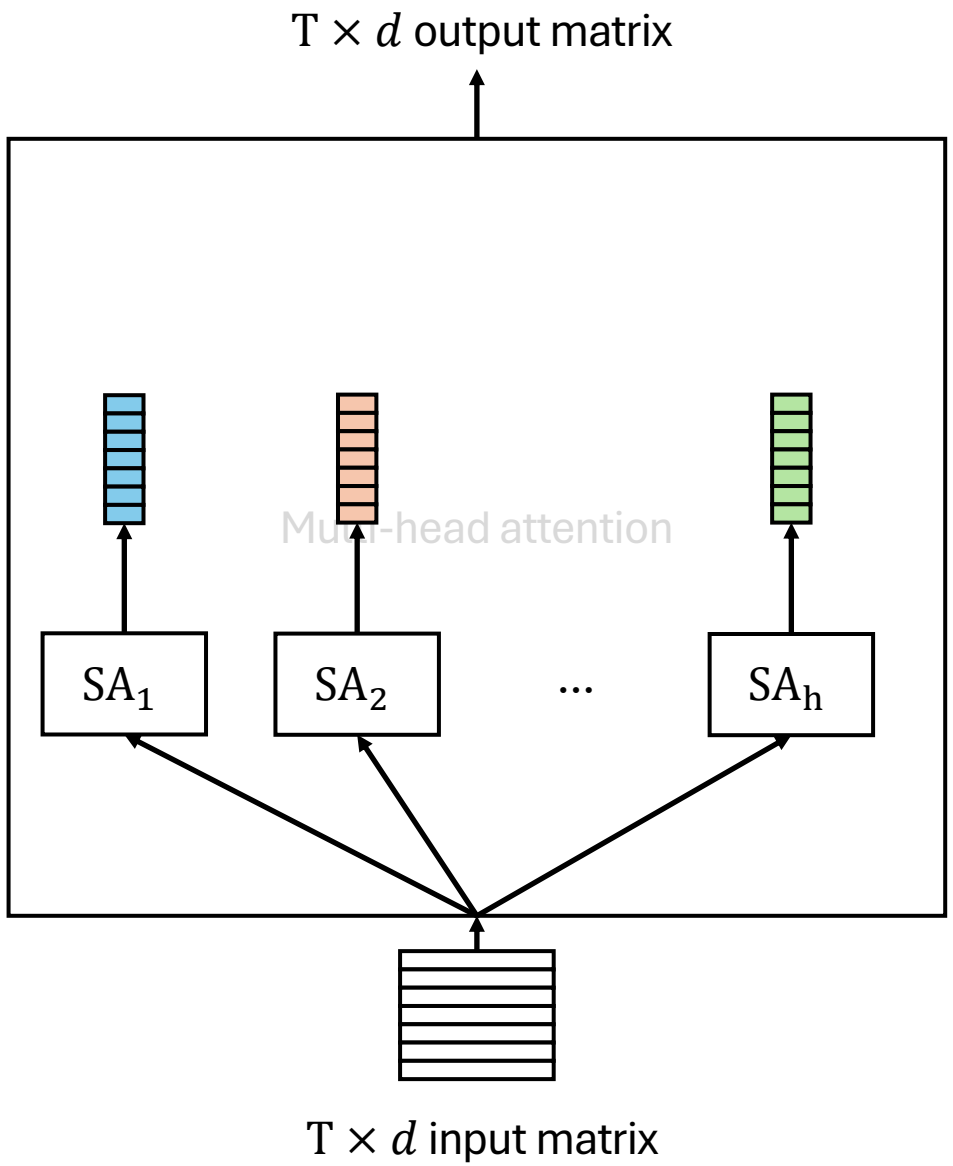

```python
def transformer_layer(X):
    X1 = layer_norm1(X + multi_head_attention(X))
    X2 = layer_norm2(X1 + fully_connected(X1))
    return X2
```

$$X_1 = \text{LayerNorm}\big(X + MHA(X)\big)$$
$$\text{Result} = \text{LayerNorm}\big(X_1 + FFN(X_1)\big)$$


T × d output matrix
Multi-head attention
T × d input matrix

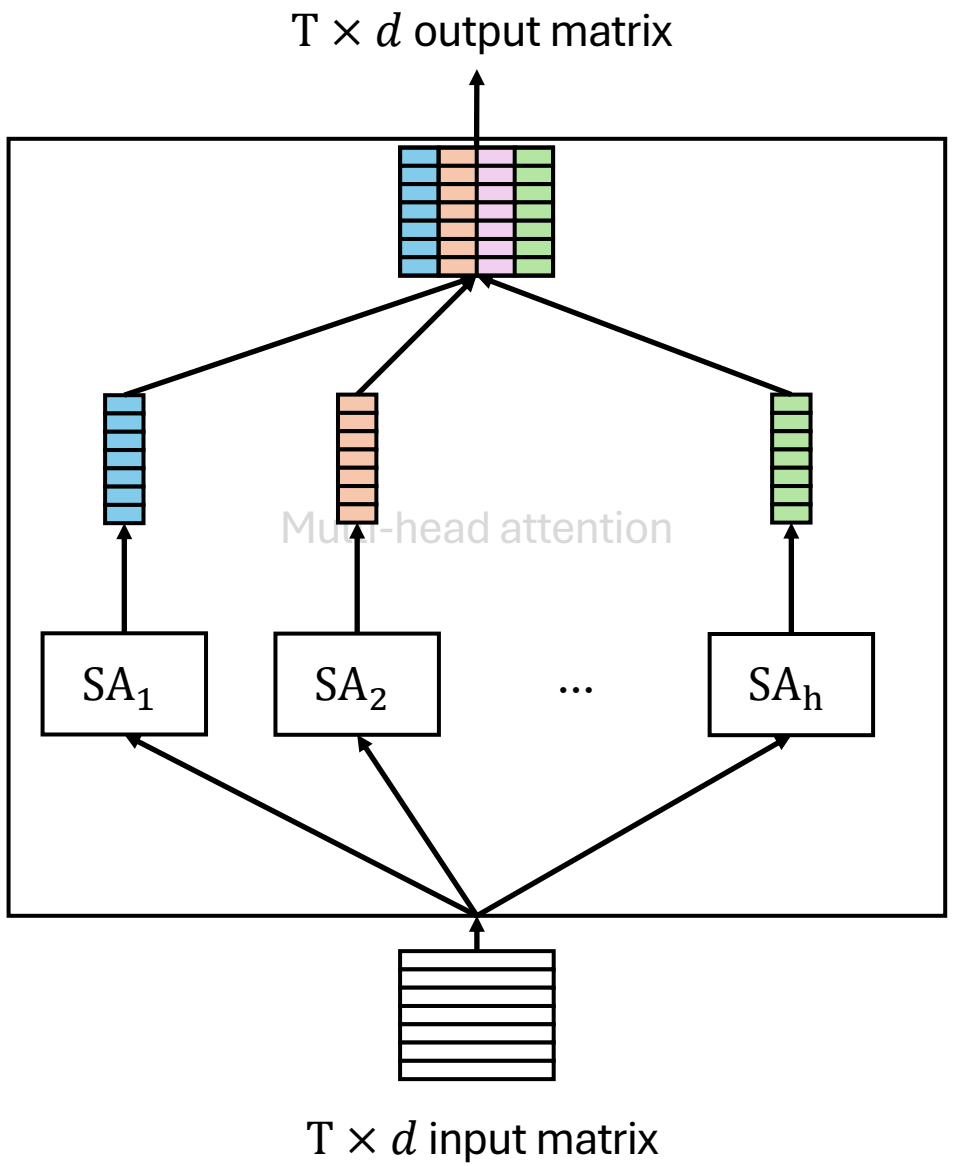


- h self-attention (SA) networks
- Analogous to channels in a CNN



Each produces a matrix of size $T \times \frac{d}{h}$

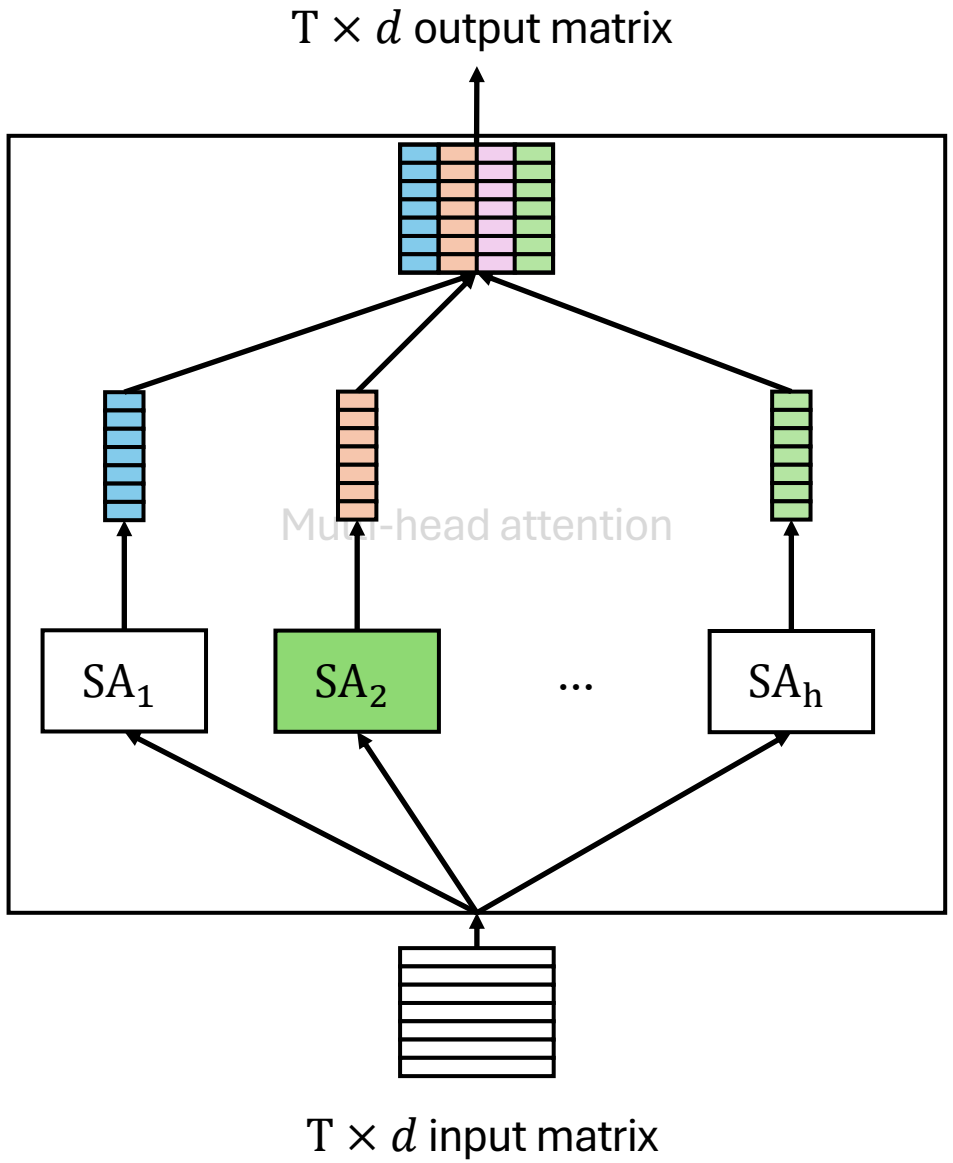
- h self-attention (SA) networks
- Analogous to channels in a CNN



These matrices are simply stacked to produce the output

Each produces a matrix of size $T \times \frac{d}{h}$

- h self-attention (SA) networks
- Analogous to channels in a CNN



These matrices are simply stacked to produce the output

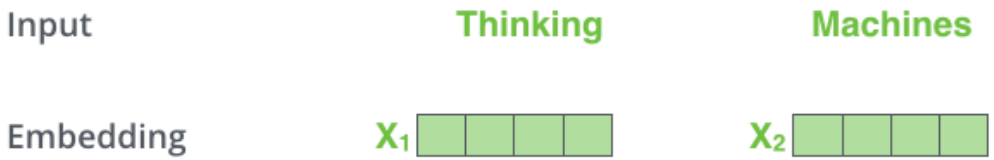
Each produces a matrix of size $T \times \frac{d}{h}$

- h self-attention (SA) networks
- Analogous to channels in a CNN

Self attention: An example

Images from https://jalammar.github.io/illustrated-transformer/

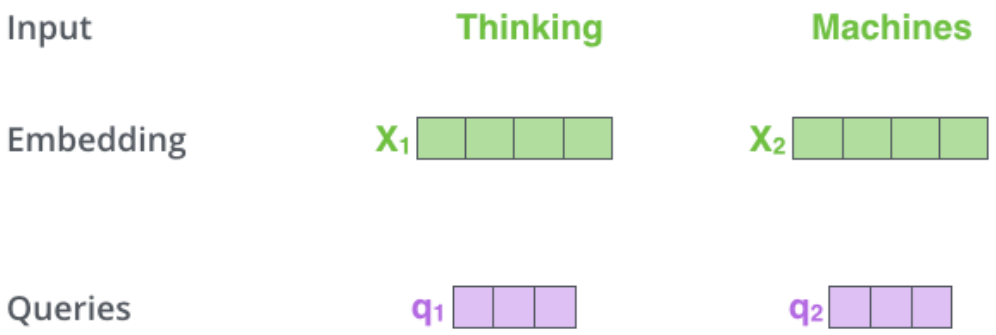
Self attention: An example



Three parameter matrices associated with this self attention block

Images from https://jalammar.github.io/illustrated-transformer/

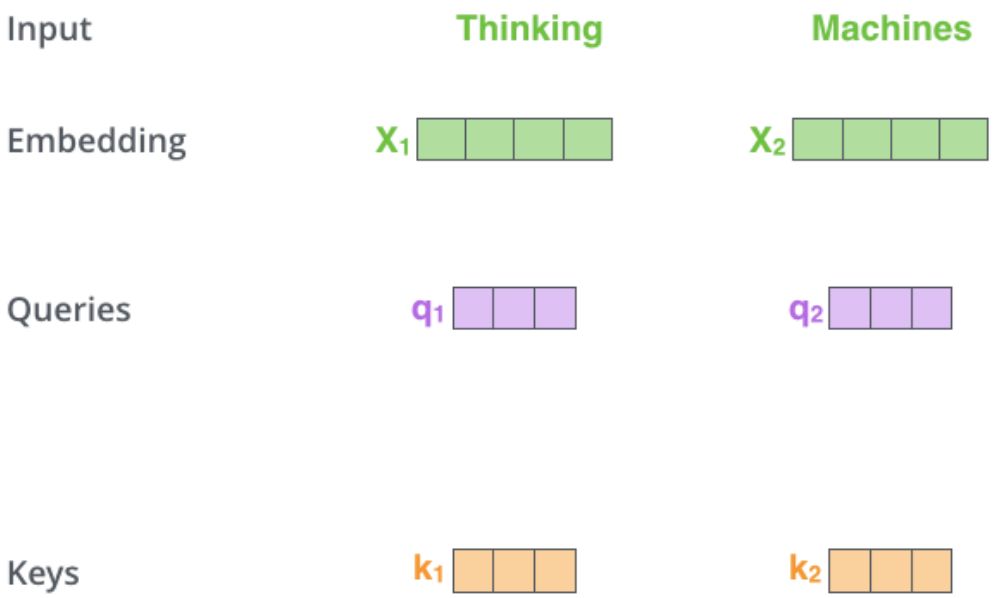
Self attention: An example

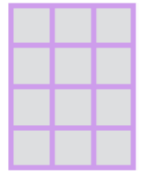


Three parameter matrices associated with this self attention block

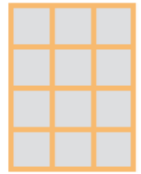
Images from https://jalammar.github.io/illustrated-transformer/

Self attention: An example

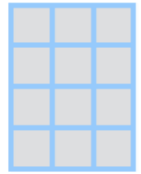




W^Q



W^K

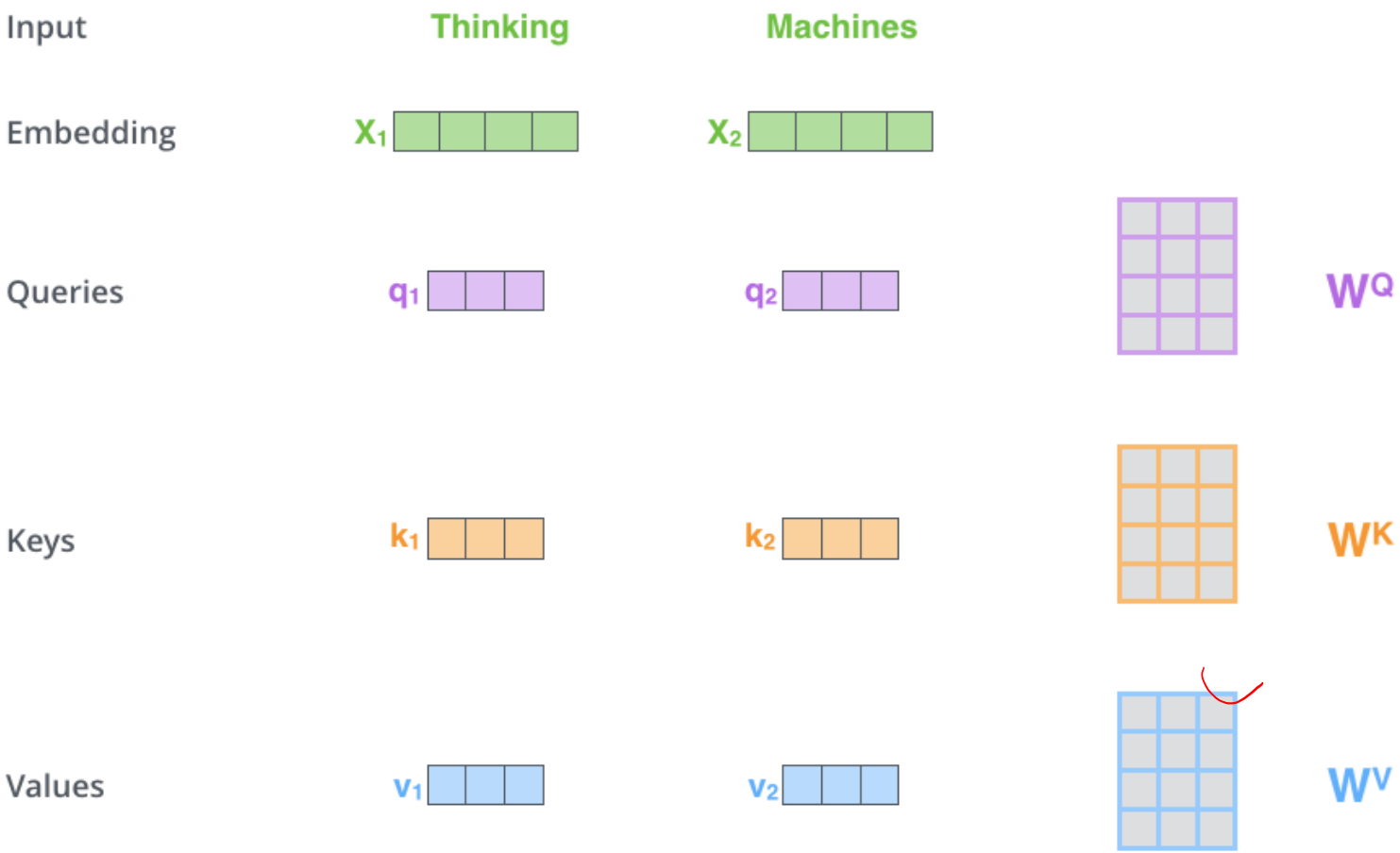


W^V

Three parameter matrices associated with this self attention block

Images from https://jalammar.github.io/illustrated-transformer/

Self attention: An example



For every element of the sequence, create three vectors that are called its *query*, *key* and *value* vectors.

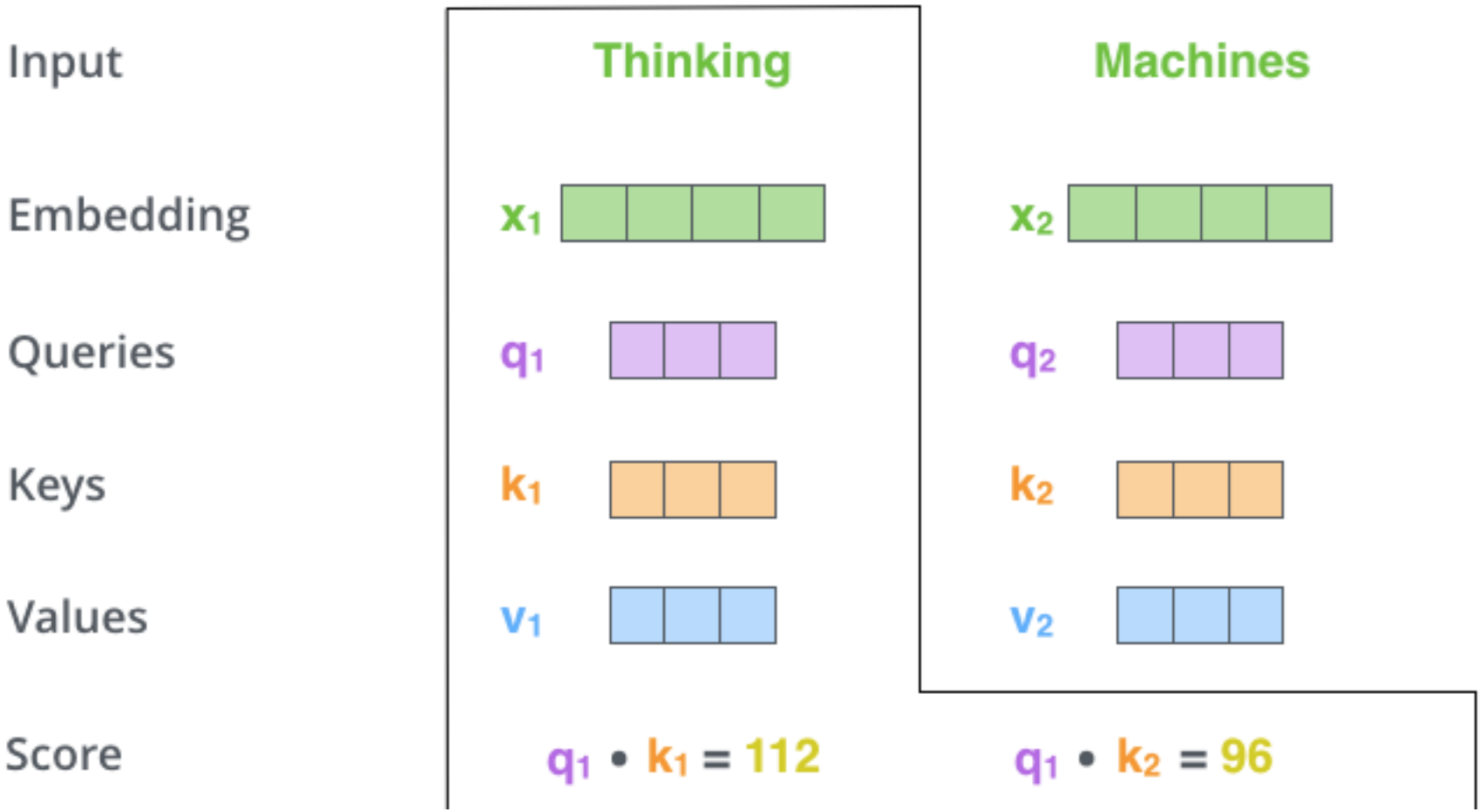
Three parameter matrices associated with this self attention block

Images from https://jalammar.github.io/illustrated-transformer/

Self attention: An example

For each word, compute the self attention.

First compute the dot product of its query vector with the key vector of all words in the sentence

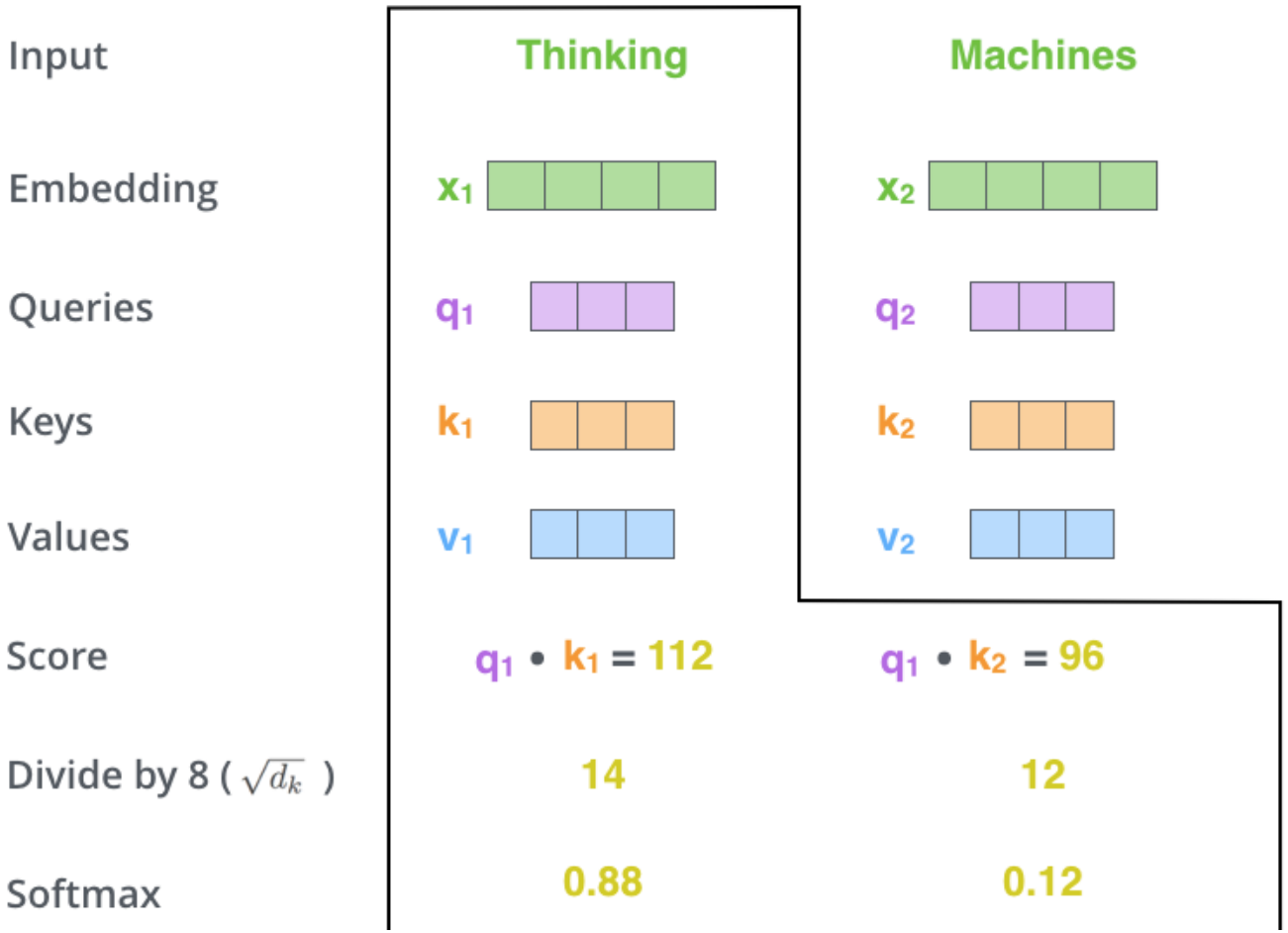


Images from https://jalammar.github.io/illustrated-transformer/

Self attention: An example

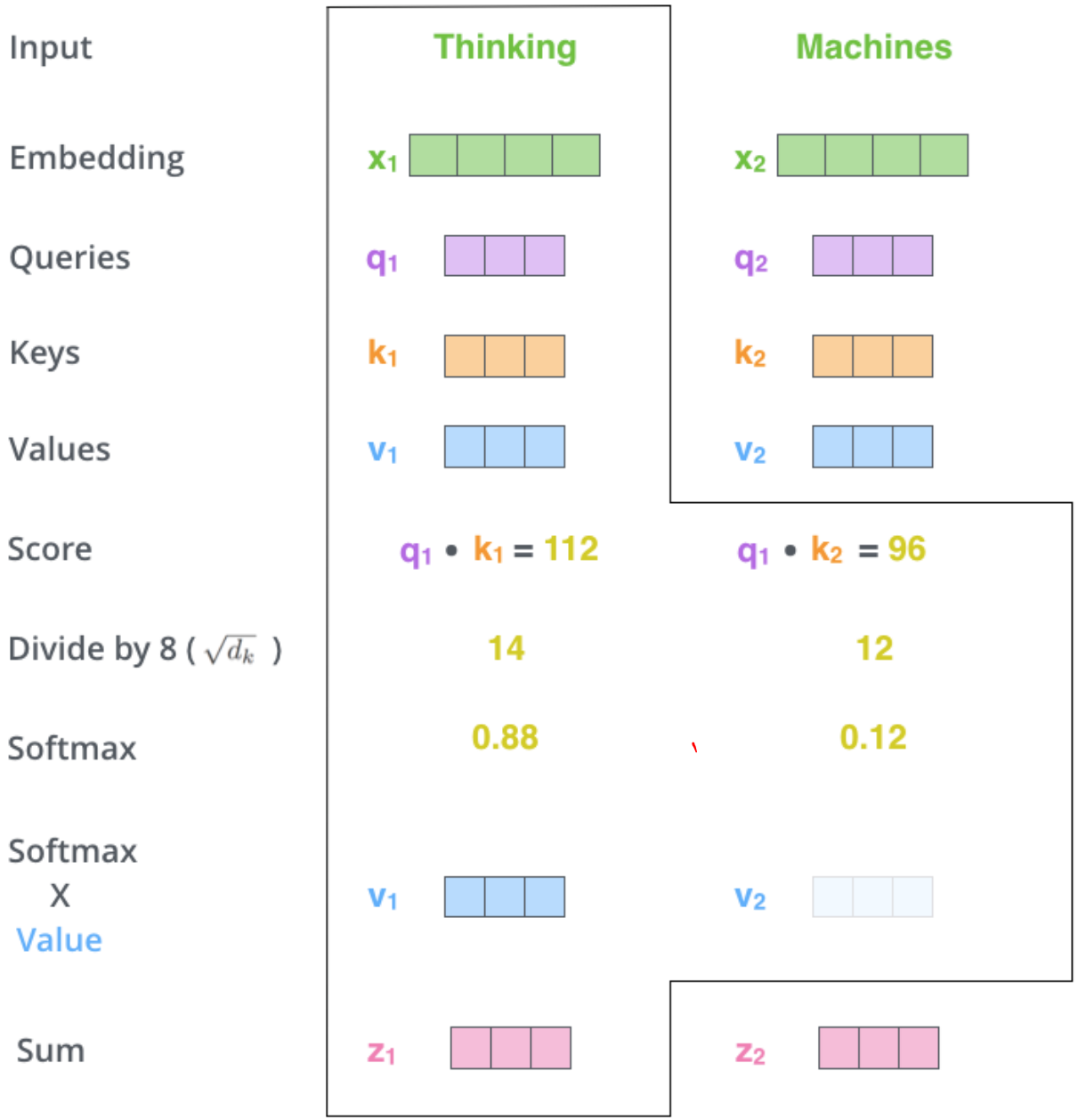
For each word, compute the self attention.

Then, normalize



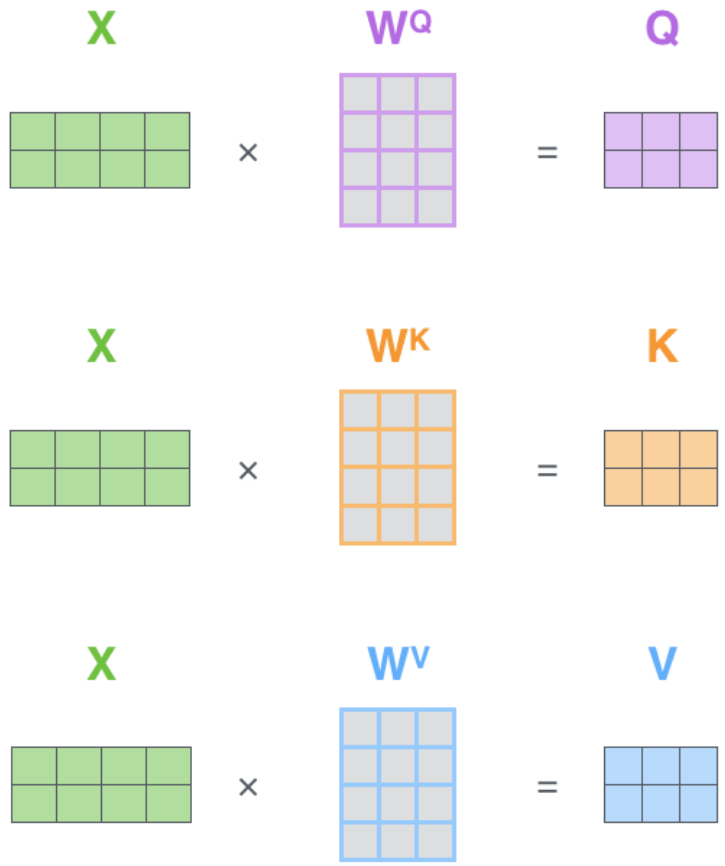
Images from https://jalammar.github.io/illustrated-transformer/

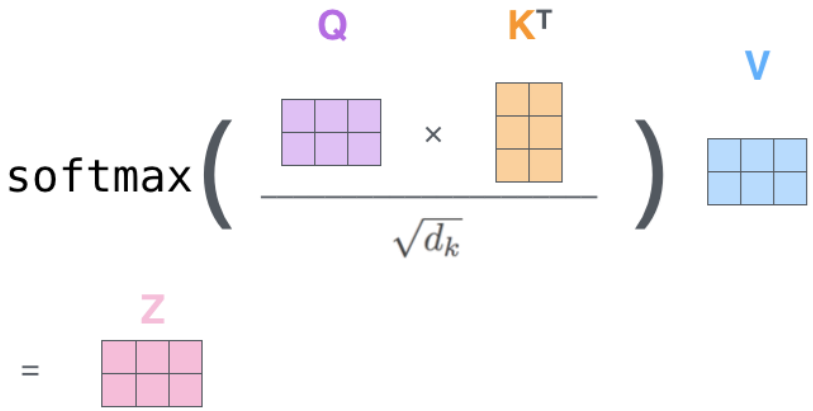
Use the attention probabilities to weigh the value vectors to produce the output vector for that word



Images from https://jalammar.github.io/illustrated-transformer/

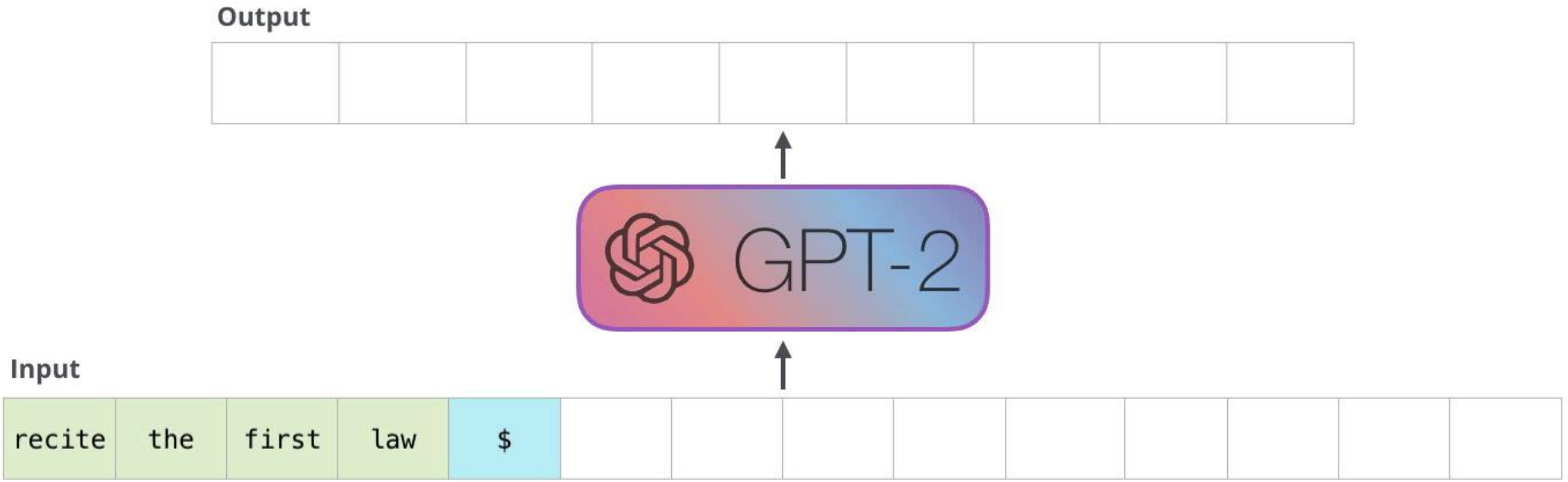
Self attention: Illustrated

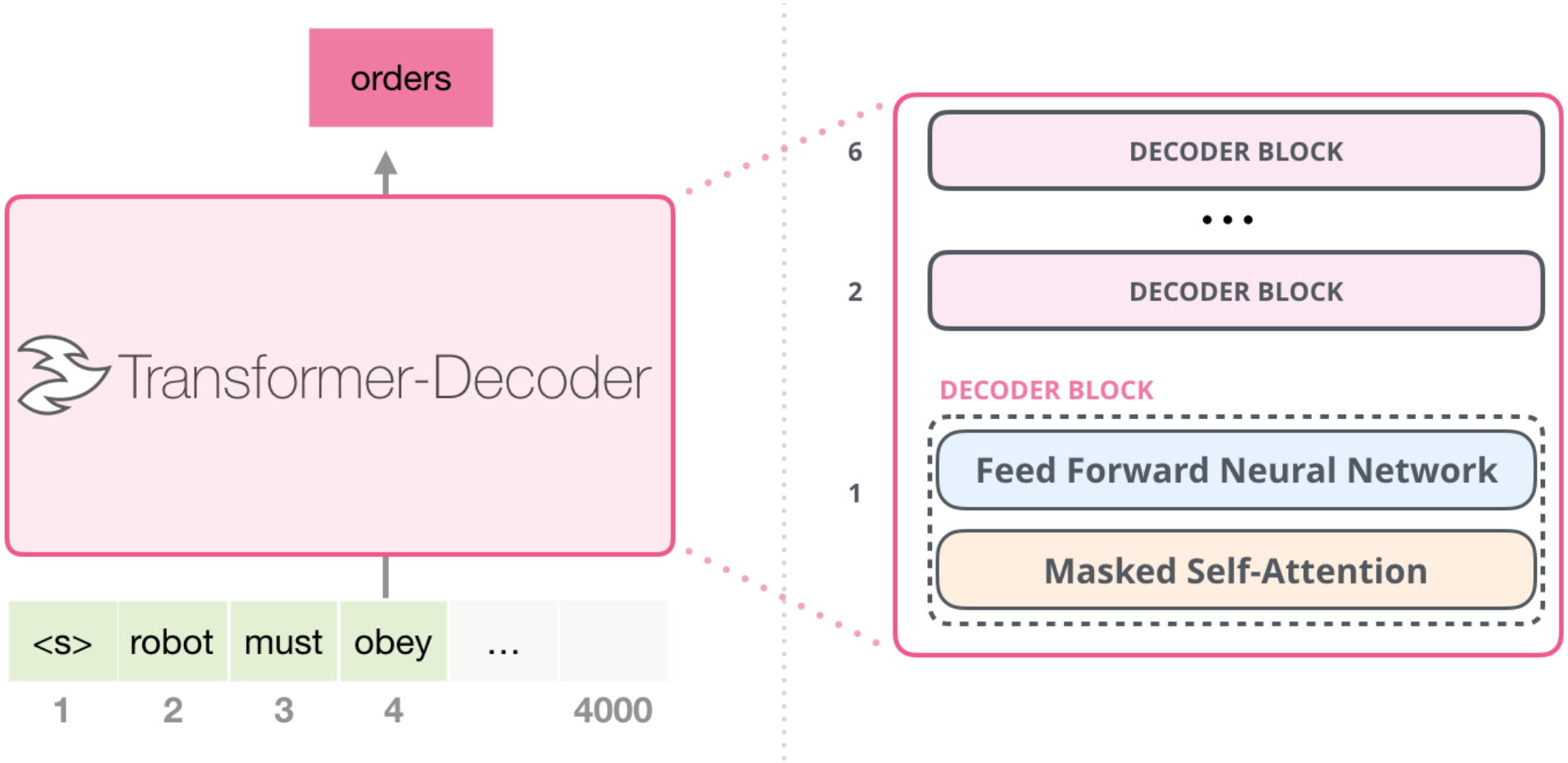


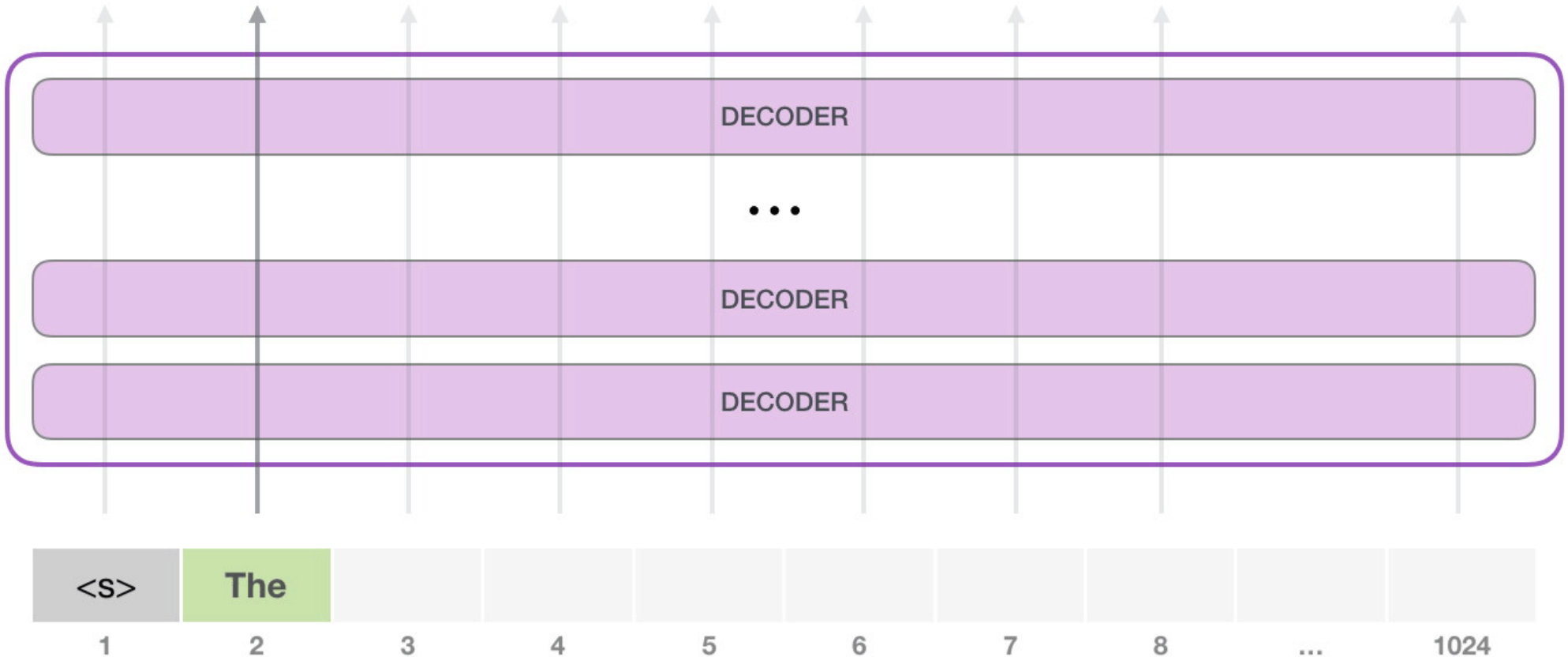


Images from https://jalammar.github.io/illustrated-transformer/

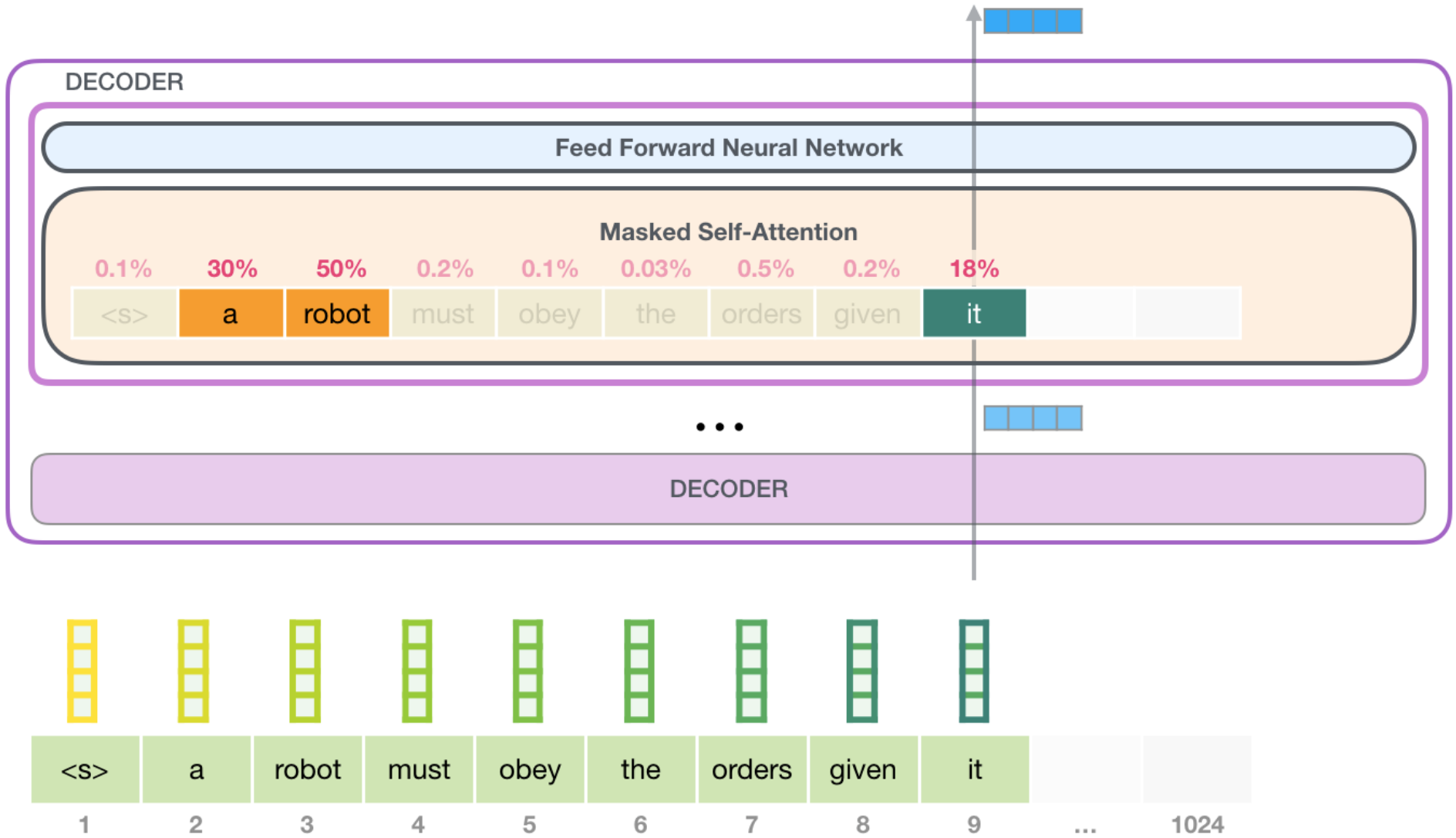
Large (Transformer-Based) Language Models

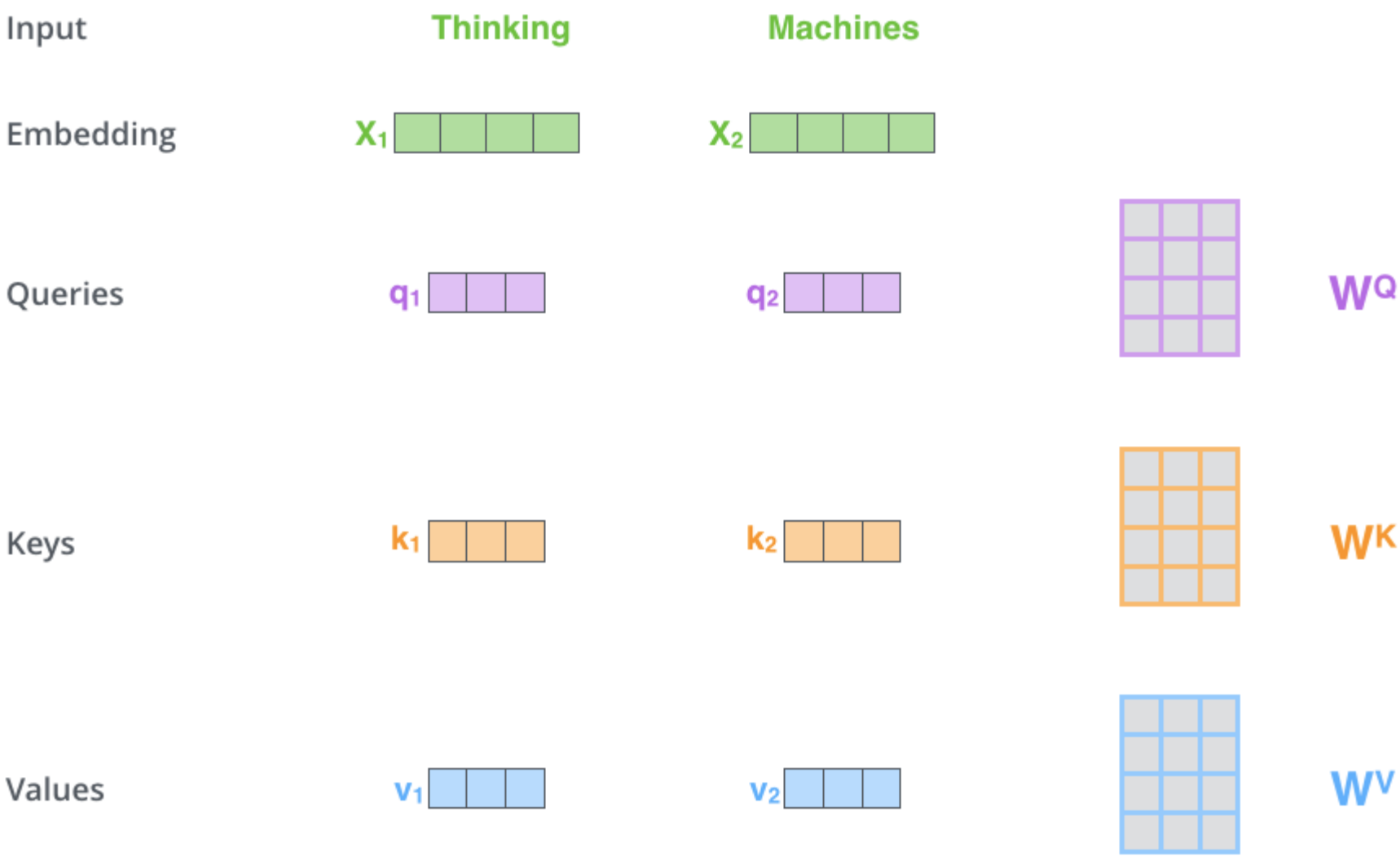



orders
Transformer-Decoder
<s>
robot
must
obey
...
1
2
3
4
4000
6
2
1
DECODER BLOCK
DECODER BLOCK
DECODER BLOCK
Feed Forward Neural Network
Masked Self-Attention

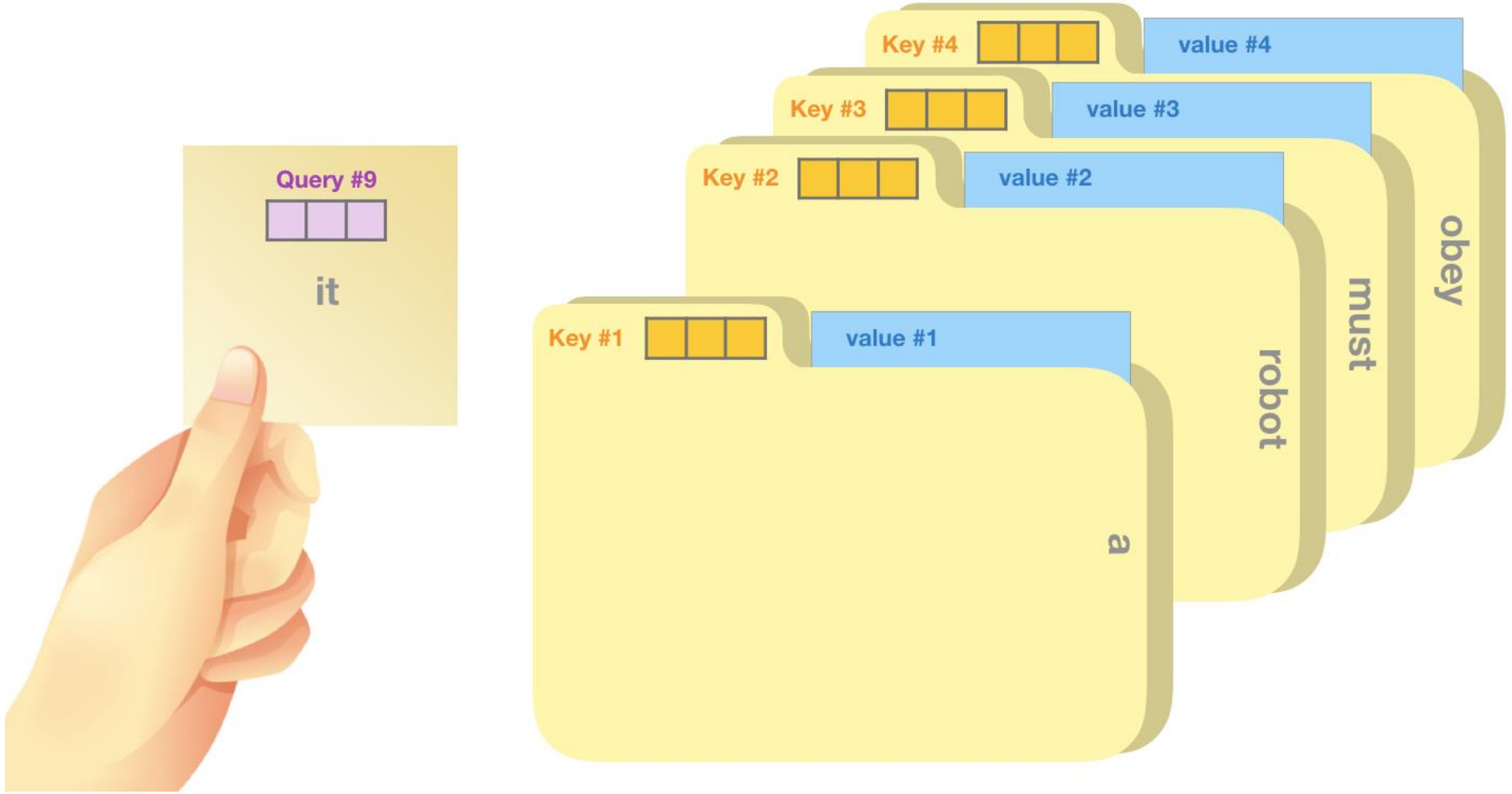

DECODER
DECODER
DECODER
<s>
The
1
2
3
4
5
6
7
8
...
1024


DECODER
• • •
DECODER
=
Positional encoding for token #1
+
Token embedding of <s>
<s>
1
2
...
1024
Token Embeddings
Positional Encodings

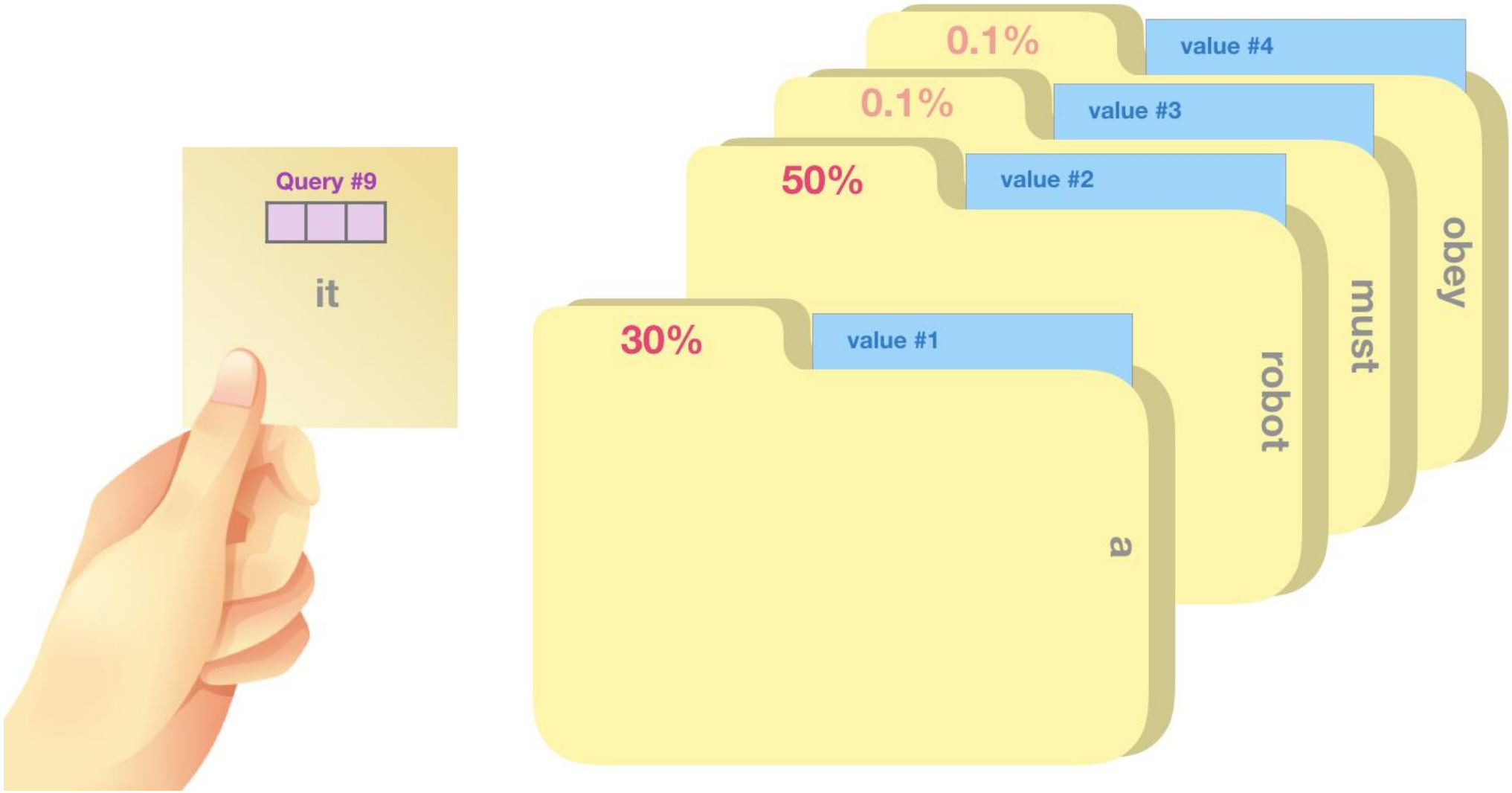

DECODER
Feed Forward Neural Network
Masked Self-Attention
0.1%
30%
50%
0.2%
0.1%
0.03%
0.5%
0.2%
18%
<s>
a
robot
must
obey
the
orders
given
it
DECODER
<s>
a
robot
must
obey
the
orders
given
it
1
2
3
4
5
6
7
8
9
...
1024

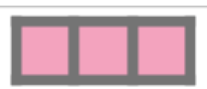

Input
Thinking
Machines
Embedding
X_1
X_2
Queries
q_1
q_2
W^Q
Keys
k_1
k_2
W^K
Values
v_1
v_2
W^V

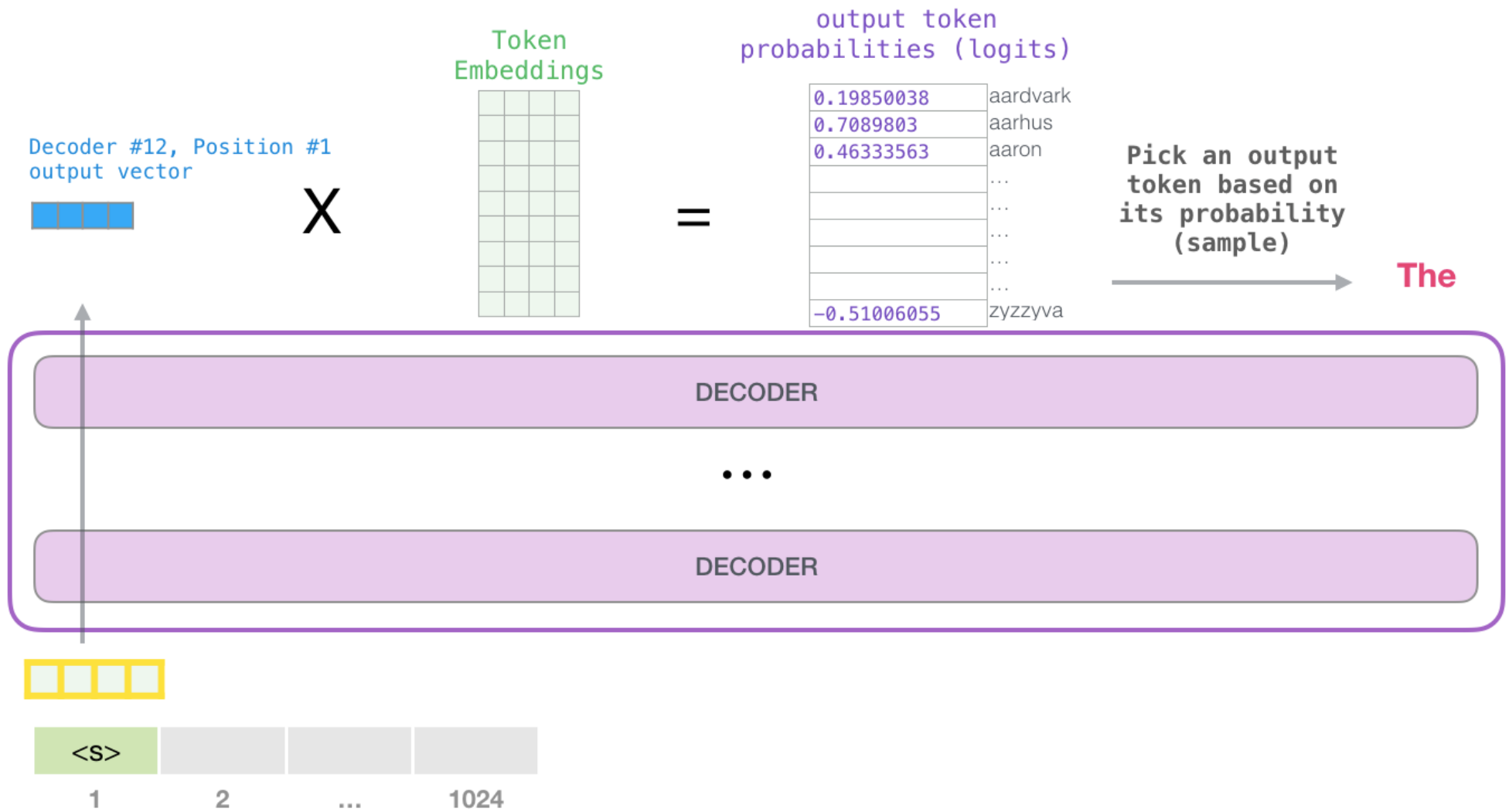
Perform dot product between query and all keys to get a raw score for each previous word (including current word).



Normalize these scores via a softmax to get a probability distribution. Then return a weighted sum of the values.



Word	Value vector	Score	Value X Score
<s>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
		Sum:	


Token
Embeddings
output token
probabilities (logits)
Decoder #12, Position #1
output vector
X
=
0.19850038
aardvark
0.7089803
aarhus
0.46333563
aaron
-0.51006055
zyzzyva
Pick an output
token based on
its probability
(sample)
The
DECODER
DECODER
<s>
1
2
...
1024

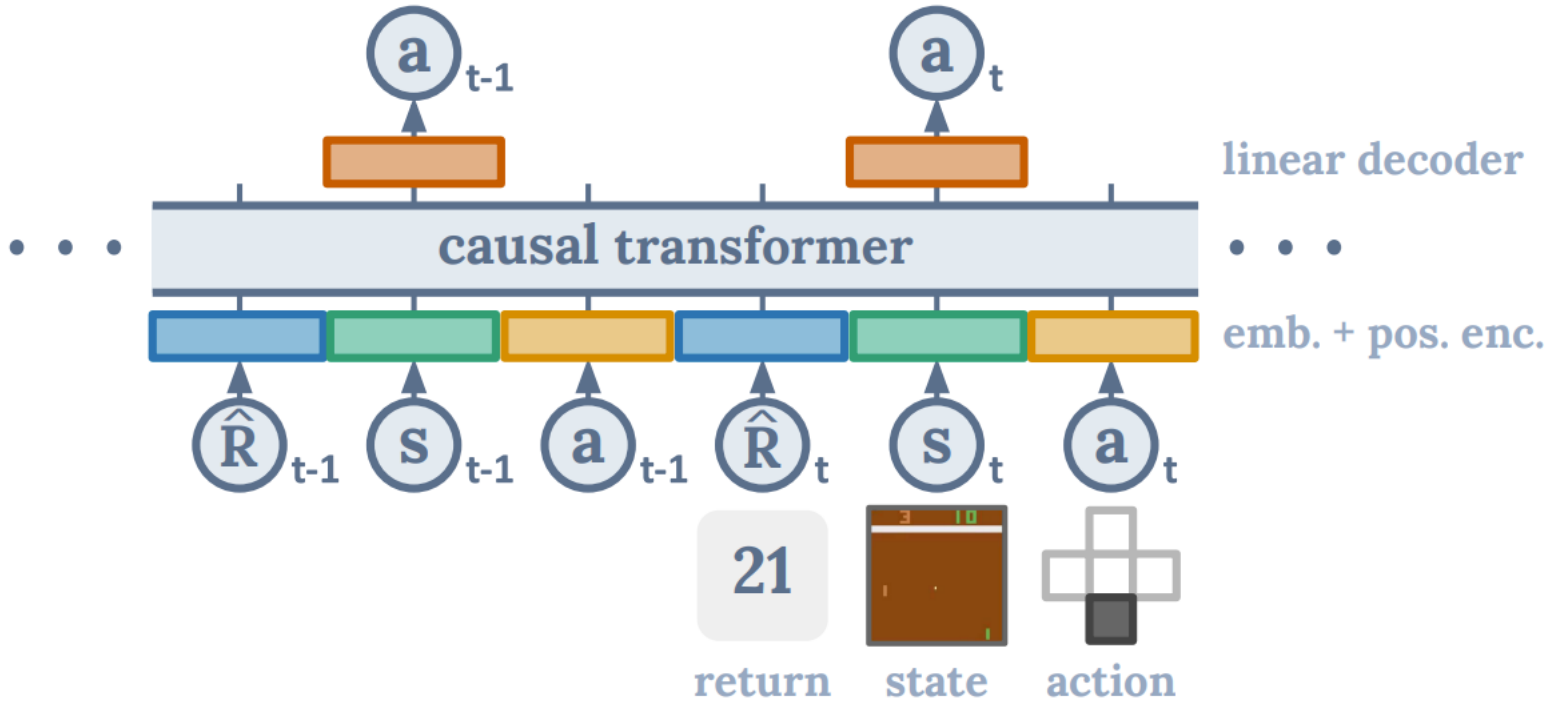
Math from the paper

- Given n inputs we create n embeddings x_i
- Map these to queries, keys, and values
- The ith output is then given by

$$z_i = \sum_{j=1}^{n} \texttt{softmax}(\{\langle q_i, k_{j'} \rangle\}_{j'=1}^{n})_j \cdot v_j$$

We hope this can learn “credit assignment” via attention
Decision Transformer uses a GPT architecture (each token only pays attention to previous tokens)

Decision Transformer



Interesting design decisions

- We want the model to output actions based on desired future returns
- Rather than (s,a,r,s') sequences

$$\tau = \left(\widehat{R}_1, s_1, a_1, \widehat{R}_2, s_2, a_2, \ldots, \widehat{R}_T, s_T, a_T\right)$$

- During training we predict actions to generate training loss
- At test time we input desired future return and starting state and then generate the action

Reinforcement Learning Upside Down: Don't Predict Rewards - Just Map Them to Actions

NNAISENSE/IDSIA Technical Report

Jürgen Schmidhuber
The Swiss AI Lab, IDSIA, USI & SUPSI
NNAISENSE, Lugano, Switzerland

23 June 2020 (based on version v1 of 5 Dec 2019)
Earlier drafts: 21 Dec, 31 Dec 2017, 20 Jan, 4 Feb, 9 Mar, 20 Apr, 16 Jul 2018

Questions

- Does the Decision Transformer make any Markov assumptions?
- How is it similar/dissimilar to behavior cloning?
- When might we expect it to perform well/poorly?

DT is closer to:

- behavior cloning + conditioning
- sequence modeling
- generative modeling

DT is *not*:

- value-based RL
- policy improvement via Bellman updates
- exploration-driven learning

Conservative Q-Learning (CQL)

- Core problem with Offline RL $\max_a Q(s,a)$

- Offline data is limited so we often overestimate OOD actions and policy exploits errors -> catastrophic failure at test time
- Core idea: pessimism in the face of uncertainty

$$\mathcal{L}_{\text{CQL}} = \underbrace{\mathcal{L}_{\text{TD}}}_{\text{Bellman error}} + \alpha \left(\underbrace{\log \sum_a \exp(Q(s,a))}_{\text{push all actions down}} - \underbrace{\mathbb{E}_{a \sim \mathcal{D}}[Q(s,a)]}_{\text{pull data actions up}} \right)$$

Implicit Q-Learning (IQL)

- Offline RL dilemma: You want to **improve the policy**, but you **can't trust values of unseen actions**

- Possible solutions
  - Regularize/penalize values of unseen actions (e.g., CQL)
  - Avoid values entirely (e.g., Decision Transformer)

- IQL
  - Do dynamic programming *without ever evaluating unseen actions*

IQL Key Ideas

- Only learn values using dataset actions
- Approximate best action via expectile regression

Mechanism:

1. Learn value function via **upper expectile**:

$$V(s) \approx \text{“high-value actions in dataset”}$$

2. Do TD updates:

$$Q(s,a) \leftarrow r + \gamma V(s')$$

3. Extract policy via **advantage-weighted BC**

Expectile Regression

- Normal regression minimizes $\mathbb{E}[(y-m)^2]$

- Expectile regression minimizes an asymmetric squared loss

$$L_\tau(u) = |\tau - \mathbf{1}(u < 0)|\, u^2$$

where:

- $u = y - m$
- $\tau \in (0,1)$ controls asymmetry

- If $\tau = 0.5$:
  - symmetric → **mean**
- If $\tau > 0.5$:
  - **penalize underestimation more**
  - → pushes estimate **upward**
- If $\tau < 0.5$:
  - pushes estimate **downward**