Attention Is All You Need

Presented by:

Shemanty Mahjabin (2105091)

and

Mahbuba Sharmin Mim (2105103)

Alice's Translation Challenge

Alice is a Data Scientist. She had been tasked with building a model to translate long, complex sentences from English to french.She was using RNNs to do it.But something was amiss.

RNNs & CNNs to the Rescue... Not Quite!

- **RNNs:**
  - Slow training due to sequential processing
  - Struggles with long-range dependencies
  - Difficulty in capturing complex relationships in long sentences
- **CNNs:**
  - Limited ability to handle sequential data
  - Cannot capture long-range dependencies effectively
  - Struggles with understanding full context of a sentence

Enter The Transformer

"I don't rely on sequential processing like RNNs."

Enter The Transformer

"I don't rely on sequential processing like RNNs."

"I use self-attention to process the entire sequence at once."

Transformer's Core Idea - Attention Mechanism

- **Self-Attention:** Models dependencies between all words in the sequence, regardless of their position.

Transformer's Core Idea - Attention Mechanism

- **Self-Attention:** Models dependencies between all words in the sequence, regardless of their position.
- **Parallel Processing:** Unlike RNNs, no sequential processing—allowing faster training.

Transformer's Core Idea - Attention Mechanism

- **Self-Attention:** Models dependencies between all words in the sequence, regardless of their position.
- **Parallel Processing:** Unlike RNNs, no sequential processing—allowing faster training.
- **Scalability:** Handles long-range dependencies better than RNNs and CNNs.

Transformer's Core Idea - Attention Mechanism

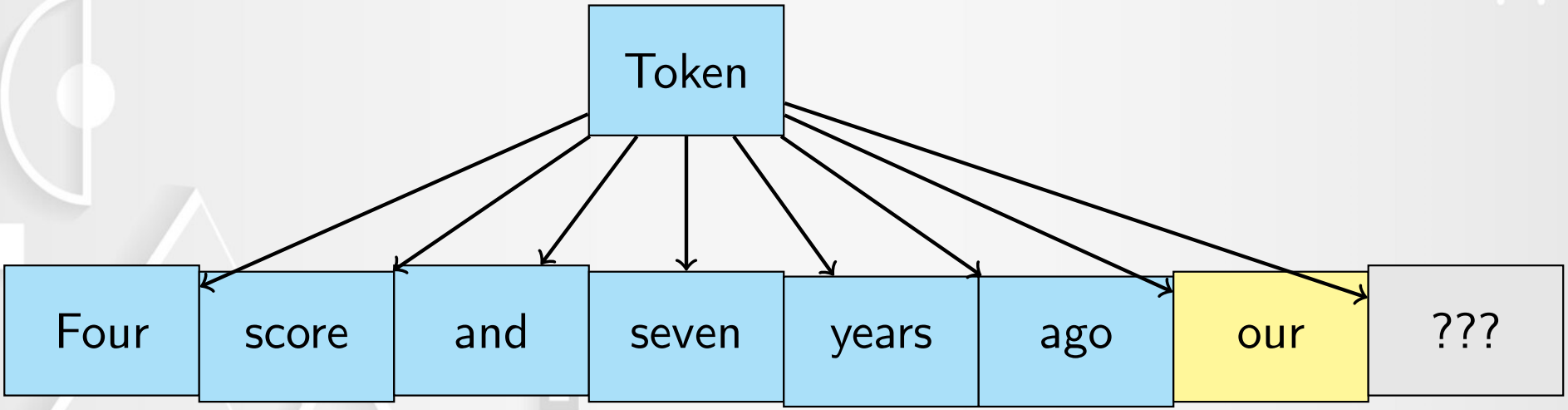
- **Self-Attention:** Models dependencies between all words in the sequence, regardless of their position.
- **Parallel Processing:** Unlike RNNs, no sequential processing—allowing faster training.
- **Scalability:** Handles long-range dependencies better than RNNs and CNNs.
- **Contextual Understanding:** Every word in the sequence can directly attend to every other word, improving context understanding.

Transformer

Goal: To take in a piece of text and predict what word comes next.

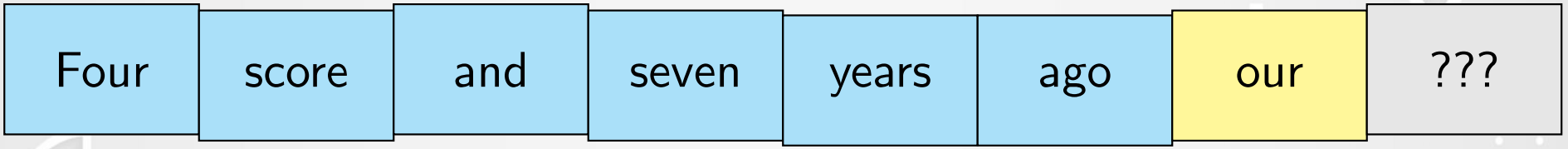
Tokens

Token: Little pieces of the input text.



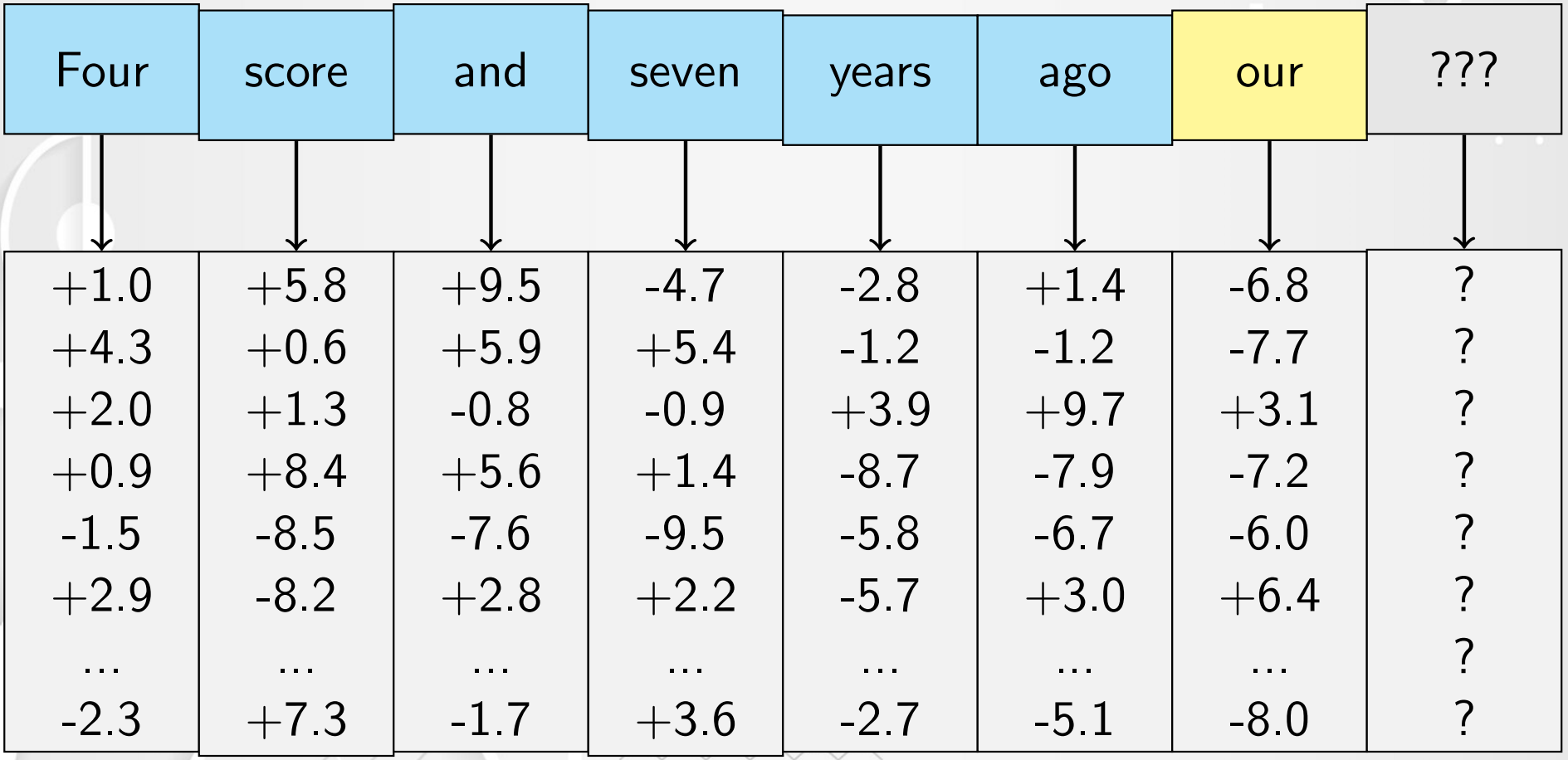
Tokens and Embeddings

Embedding: Associating each token with a high-dimensional vector.

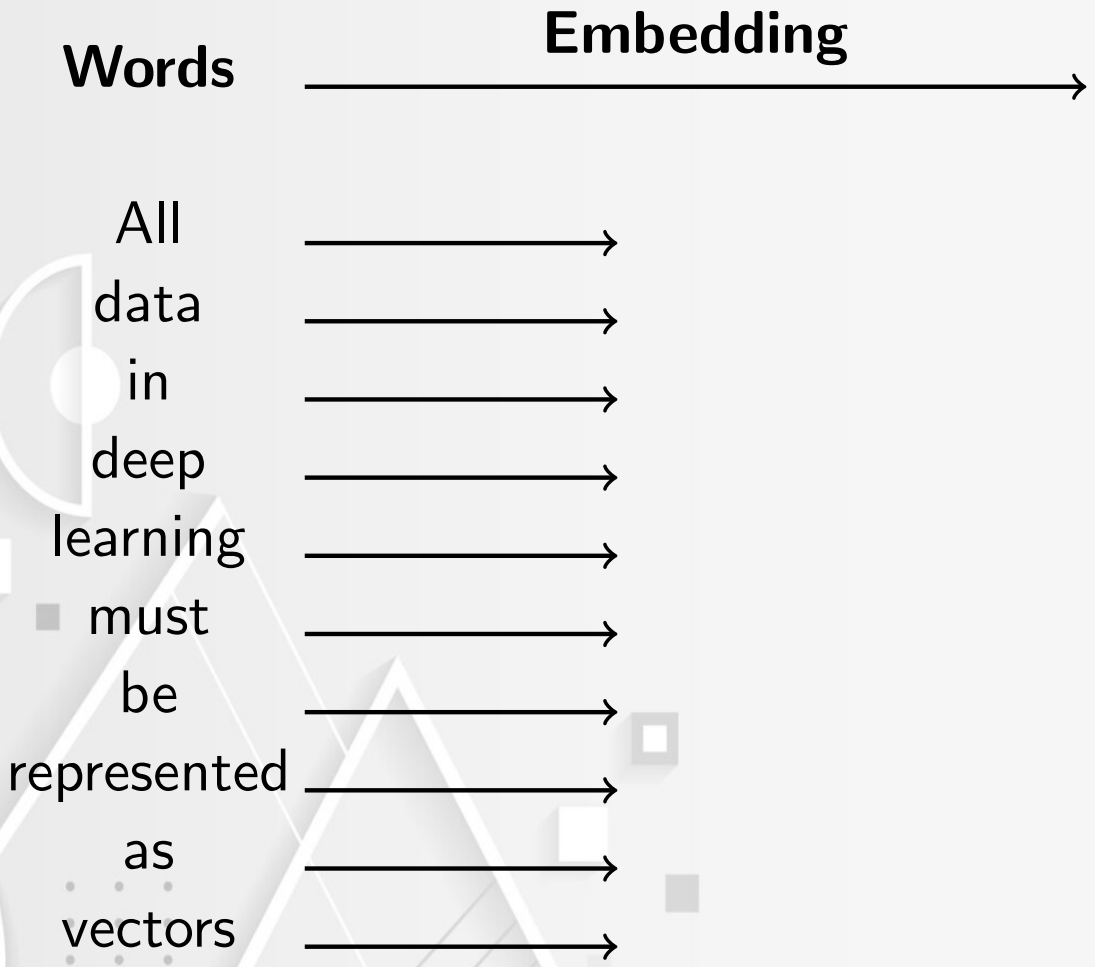


Tokens and Embeddings

Embedding: Associating each token with a high-dimensional vector.



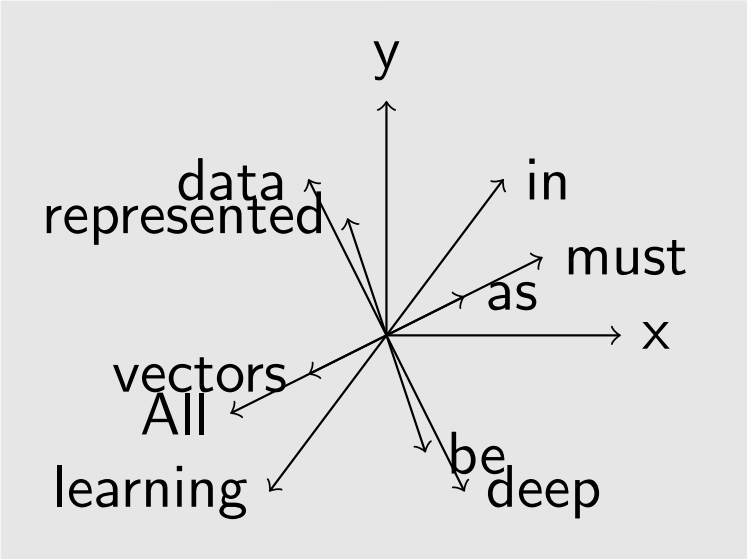
Attention in Transformers



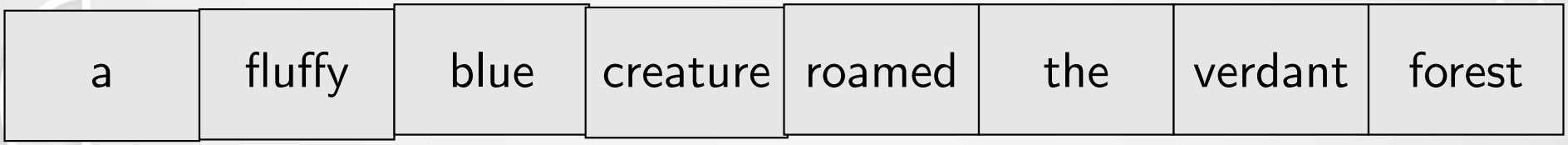
Attention in Transformers

Words —**Embedding**→ **Vectors**

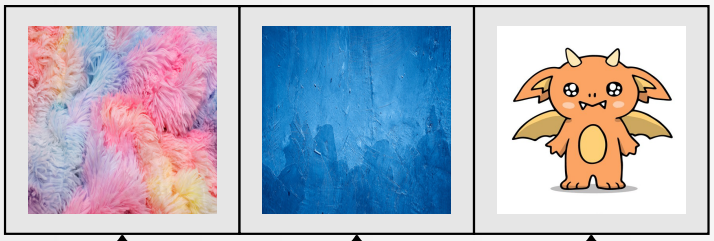
All →
data →
in →
deep →
learning →
must →
be →
represented →
as →
vectors →



Attention in Transformers



Attention in Transformers



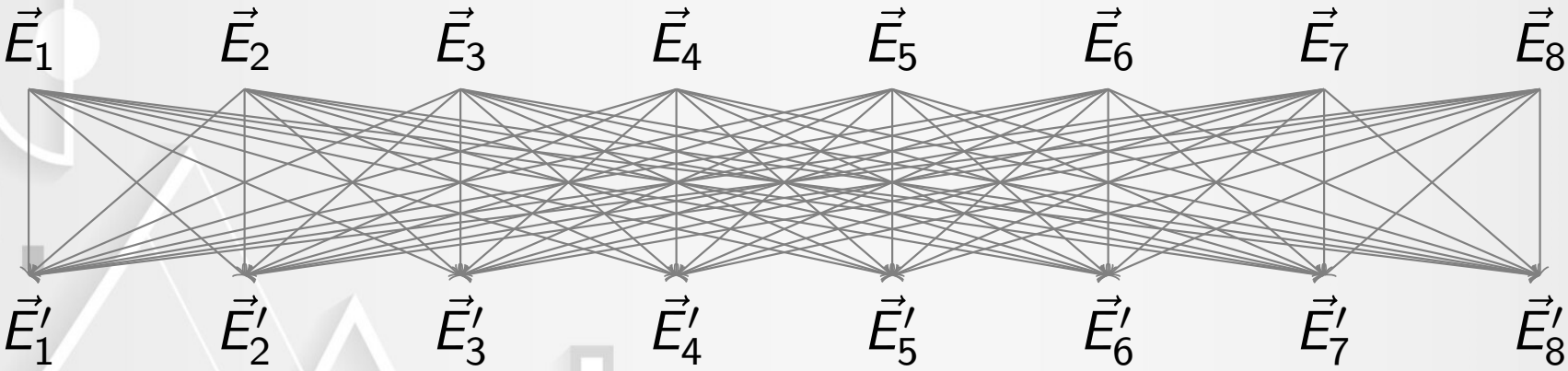

a	fluffy	blue	creature	roamed	the	verdant	forest
6.0	5.6	8.8	1.6	4.5	5.2	0.3	7.2
0.2	5.8	8.0	6.1	7.1	0.5	1.6	3.1
3.0	6.5	7.0	1.2	9.7	2.0	6.2	2.1
6.5	5.7	9.1	8.4	8.6	0.2	5.7	3.9
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
3.0	3.6	8.6	6.9	1.7	7.0	6.8	2.3



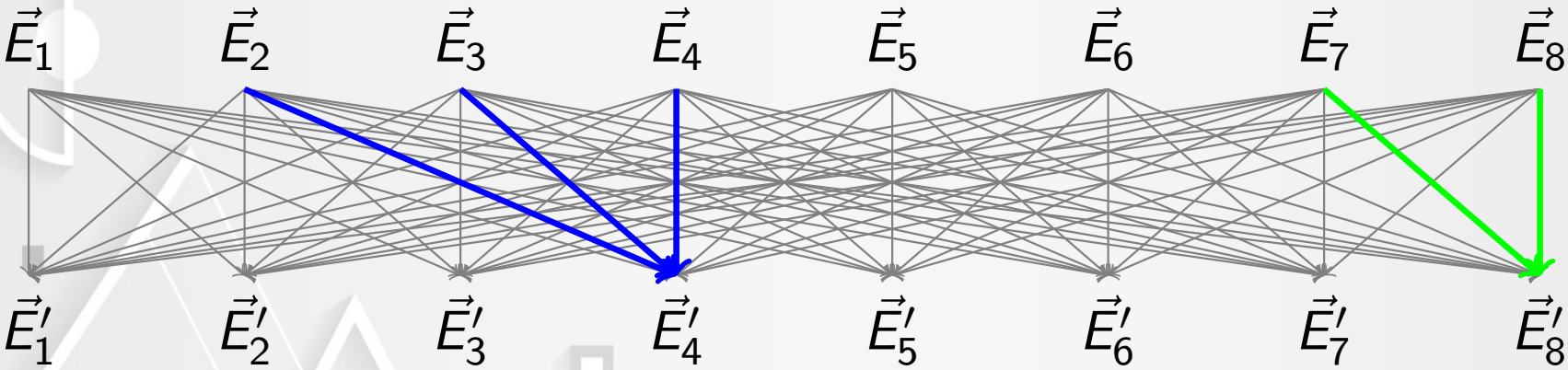
a	fluffy	blue	creature	roamed	the	verdant	forest
$\vec{E}_1$	$\vec{E}_2$	$\vec{E}_3$	$\vec{E}_4$	$\vec{E}_5$	$\vec{E}_6$	$\vec{E}_7$	$\vec{E}_8$
$\vec{E}'_1$	$\vec{E}'_2$	$\vec{E}'_3$	$\vec{E}'_4$	$\vec{E}'_5$	$\vec{E}'_6$	$\vec{E}'_7$	$\vec{E}'_8$



a fluffy blue creature roamed the verdant forest



a fluffy blue creature roamed the verdant forest



High-Level Architecture

Transformer consists of -

- encoder block
- decoder block

Encoder Components

A simple encoder consist of 2 parts :

- **Self-Attention:** Relates each word to every other word in the sentence.

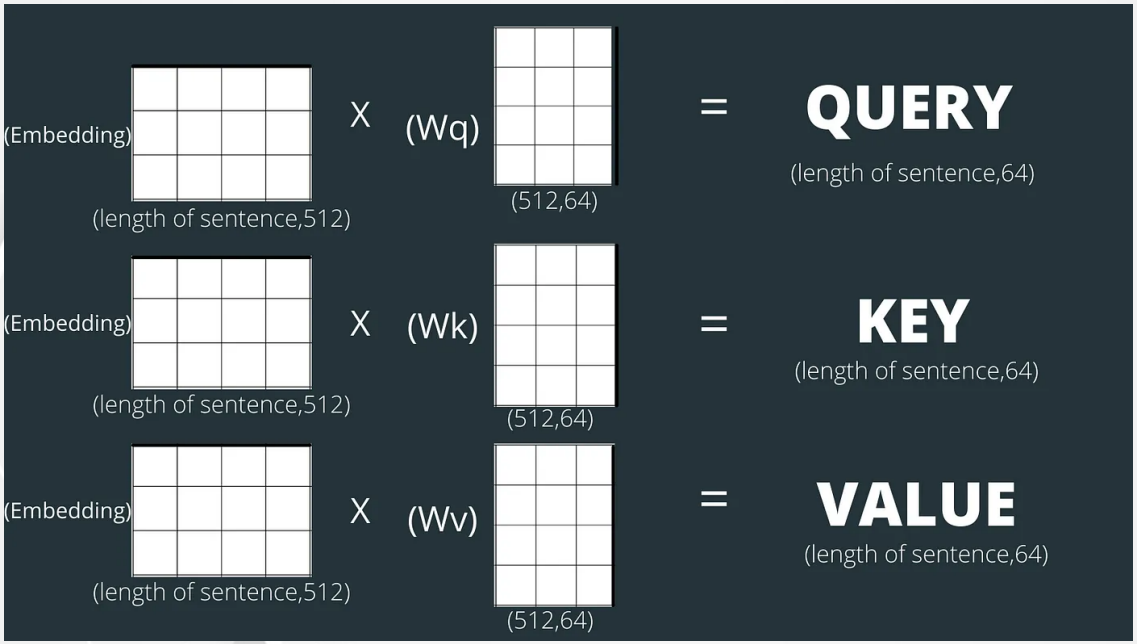
Encoder Components

A simple encoder consist of 2 parts :

- **Self-Attention:** Relates each word to every other word in the sentence.
- **Feed-Forward Neural Network:** The output of the self-attention mechanism is passed as input to this feed-forward network.

Self-Attention in Transformers

- **Key Steps:**
  - Create 3 vectors: **Query, Key, Value**.
  - **Dimensions:** 64-dimensional vectors.
  - **Matrix Multiplication:** $Q = W_q \times \text{Embedding}$, $K = W_k \times \text{Embedding}$, $V = W_v \times \text{Embedding}$.
  - **Weight Matrices:** W_q, W_k, W_v of size $(512, 64)$.



Self-Attention: Step 1 - Calculate Score

- **Score Calculation:** Measures the relationship between words.
- **Dot Product:** Compute the dot product of **Query** q_1 and **Key** k for each word.
- **Purpose:** Determines how much each word is related to others in the sentence.

Self-Attention: Step 1 - Calculate Score

- **Score Calculation:** Measures the relationship between words.
- **Dot Product:** Compute the dot product of **Query** q_1 and **Key** k for each word.
- **Purpose:** Determines how much each word is related to others in the sentence.

Inputs	*je*	*suis*	*étudiant*
Embedding	x_1	x_2	x_3
Query	q_1	q_2	q_3
Key	k_1	k_2	k_3
Value	v_1	v_2	v_3
Score	$q_1 \cdot k_1 = 146$	$q_1 \cdot k_2 = 87$	$q_1 \cdot k_3 = 24$

Self-Attention: Steps 2 3

- **Step 2: Scale the Scores**
  - Divide by $\sqrt{d_k}$ (dimension of key vectors).
  - **Purpose:** Ensures more stable gradients.
- **Step 3: Softmax of Scores**
  - Apply softmax to scores.
  - **Result:** Scores sum to 1.
  - **Purpose:** Determines how much each word contributes to the output.

Inputs	*je*	*suis*	*étudiant*
Embedding	x_1	x_2	x_3
Query	q_1	q_2	q_3
Key	k_1	k_2	k_3
Value	v_1	v_2	v_3
Score	$q_1 \cdot k_1 = 146$	$q_1 \cdot k_2 = 87$	$q_1 \cdot k_3 = 24$

Self-Attention: Steps 2 3

- **Step 2: Scale the Scores**
  - Divide by $\sqrt{d_k}$ (dimension of key vectors).
  - **Purpose:** Ensures more stable gradients.
- **Step 3: Softmax of Scores**
  - Apply softmax to scores.
  - **Result:** Scores sum to 1.
  - **Purpose:** Determines how much each word contributes to the output.

Inputs	*je*	*suis*	*étudiant*
Embedding	x_1	x_2	x_3
Query	q_1	q_2	q_3
Key	k_1	k_2	k_3
Value	v_1	v_2	v_3
Score	$q_1 \cdot k_1 = 146$	$q_1 \cdot k_2 = 87$	$q_1 \cdot k_3 = 24$
Divide by 8	18.25	10.875	3
Softmax	0.82	0.12	0.06

Self-Attention: Steps 4 5

- **Step 4: Multiply Softmax by Value**
  - Multiply softmax scores with **Value** vector v for each word.
- **Step 5: Sum to Produce Output**
  - Add the results to produce a single 64-dimensional output vector.

Inputs	*je*	*suis*	*étudiant*
Embedding	x_1	x_2	x_3
Query	q_1	q_2	q_3
Key	k_1	k_2	k_3
Value	v_1	v_2	v_3
Score	$q_1 \cdot k_1 = 146$	$q_1 \cdot k_2 = 87$	$q_1 \cdot k_3 = 24$
Divide by 8	18.25	10.875	3
Softmax	0.82	0.12	0.06

Self-Attention: Steps 4 5

- **Step 4: Multiply Softmax by Value**
  - Multiply softmax scores with **Value** vector v for each word.
- **Step 5: Sum to Produce Output**
  - Add the results to produce a single 64-dimensional output vector.

Inputs	*je*	*suis*	*étudiant*
Embedding	x_1	x_2	x_3
Query	q_1	q_2	q_3
Key	k_1	k_2	k_3
Value	v_1	v_2	v_3
Score	$q_1 \cdot k_1 = 146$	$q_1 \cdot k_2 = 87$	$q_1 \cdot k_3 = 24$
Divide by 8	18.25	10.875	3
Softmax	0.82	0.12	0.06
softmax $\times$ value	$0.82 \cdot v_1$	$0.12 \cdot v_2$	$0.06 \cdot v_3$

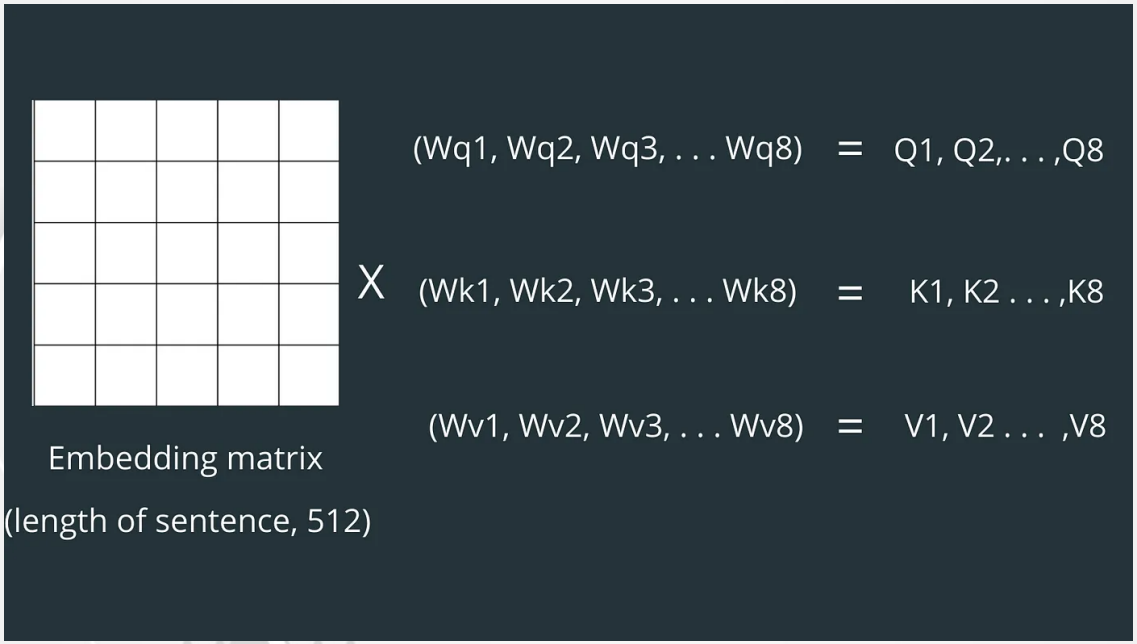
$$z_1 = \sum 0.86 \cdot v_1 + 0.12 \cdot v_2 + 0.06 \cdot v_3$$

Multi-Head Attention

- **Single-Head Attention:** One weight matrix (Wq, Wk, Wv) for query, key, and value.
- **Multi-Head Attention:** Multiple weight matrices for producing multiple queries, keys, and values.
- **Process:**
  - Repeat the self-attention operations multiple times for each head.
  - Combine the results from each head.
- **In Transformer:** 8 Multi-Head Attention used.

Multi-Head Attention

- **Single-Head Attention:** One weight matrix (Wq, Wk, Wv) for query, key, and value.
- **Multi-Head Attention:** Multiple weight matrices for producing multiple queries, keys, and values.
- **Process:**
  - Repeat the self-attention operations multiple times for each head.
  - Combine the results from each head.
- **In Transformer:** 8 Multi-Head Attention used.

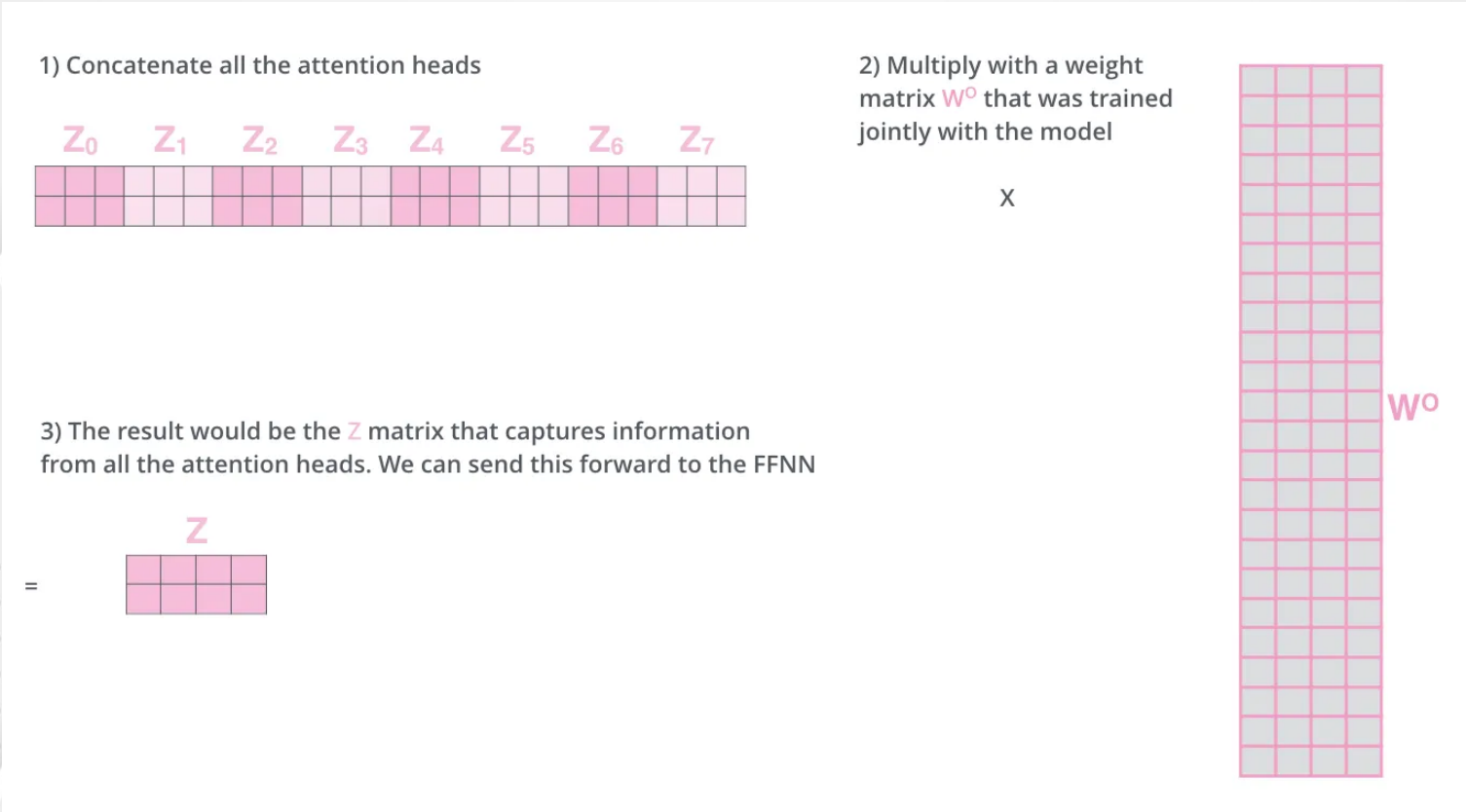


Attention Heads and Final Output

- **For Each Head:**
  - One Z matrix (attention head) per query, key, and value matrix.
- **Total Heads:** 8 attention heads in the Transformer.
- **Final Output:**
  - Concatenate all attention heads.
  - Multiply with other weight matrices to get the final Z matrix.

Attention Heads and Final Output

- **For Each Head:**
  - One Z matrix (attention head) per query, key, and value matrix.
- **Total Heads:** 8 attention heads in the Transformer.
- **Final Output:**
  - Concatenate all attention heads.
  - Multiply with other weight matrices to get the final Z matrix.



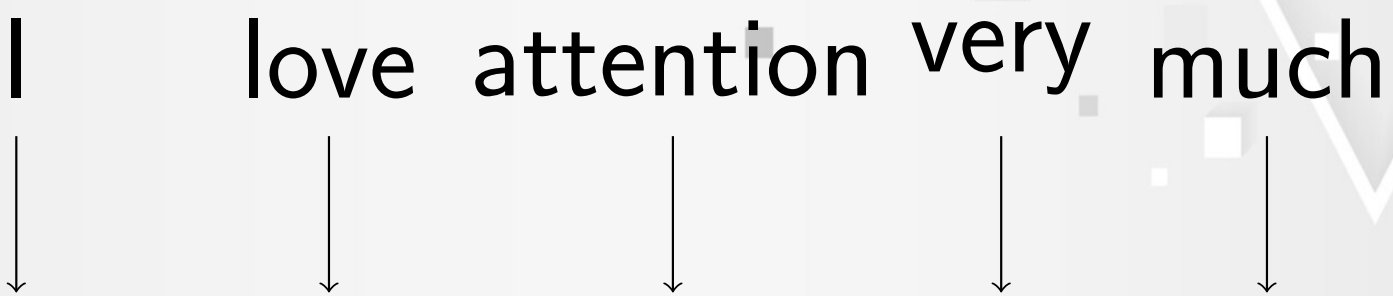
Encoder Layer - Self-Attention

Encoder Input

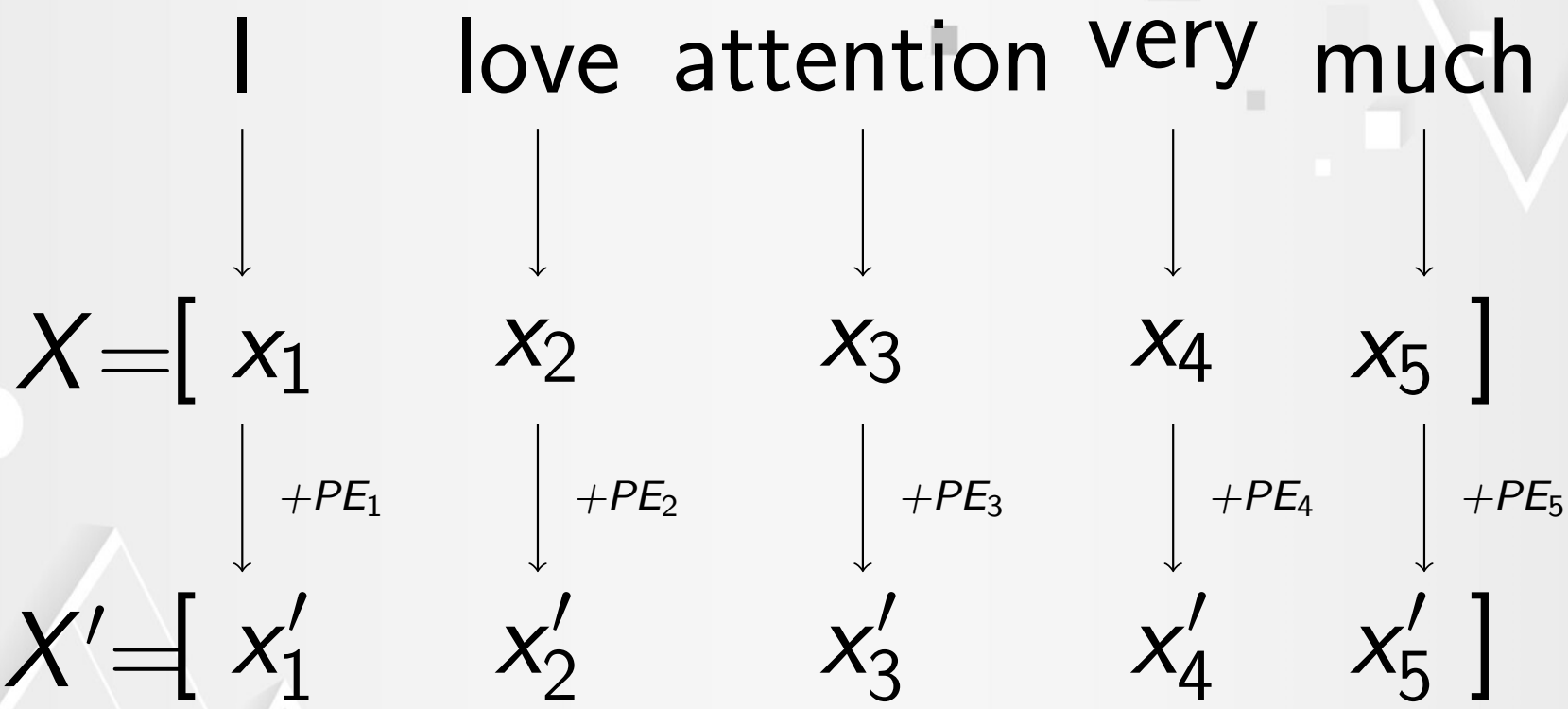
Encoder Input

I love attention very much

Encoder Input



Encoder Input



Encoder Output

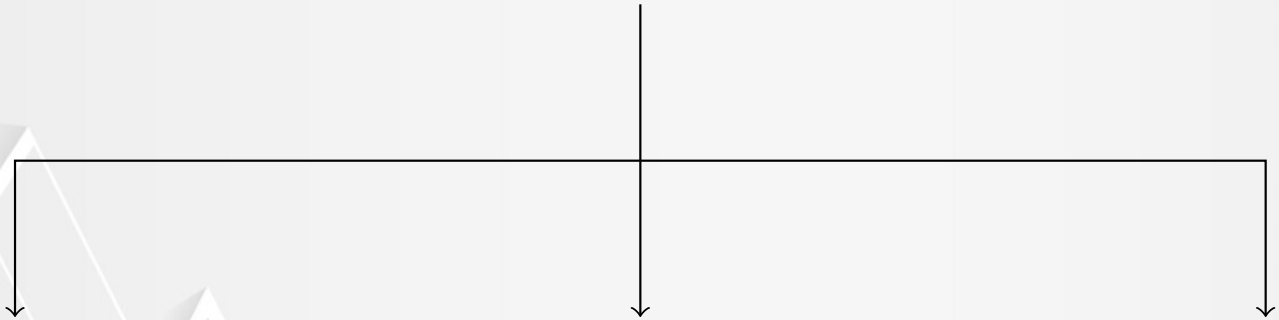
$$X'$$

$$\begin{bmatrix} x'_{11} & x'_{12} & \cdots & \cdots \\ x'_{21} & x'_{22} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Encoder Output

$$X'$$

$$\begin{bmatrix} x'_{11} & x'_{12} & \cdots & \cdots \\ x'_{21} & x'_{22} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

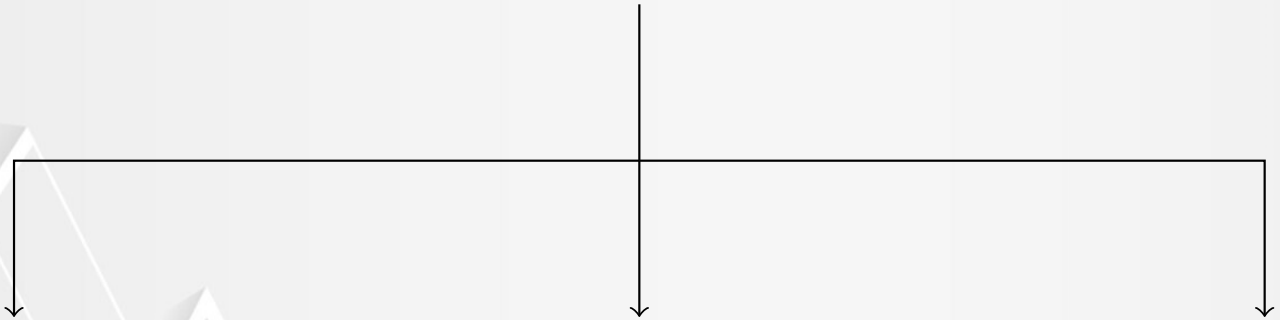


Encoder Output

$$X'$$

$$\begin{bmatrix} x'_{11} & x'_{12} & \cdots & \cdots \\ x'_{21} & x'_{22} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

(seq, 512)



$$Q$$

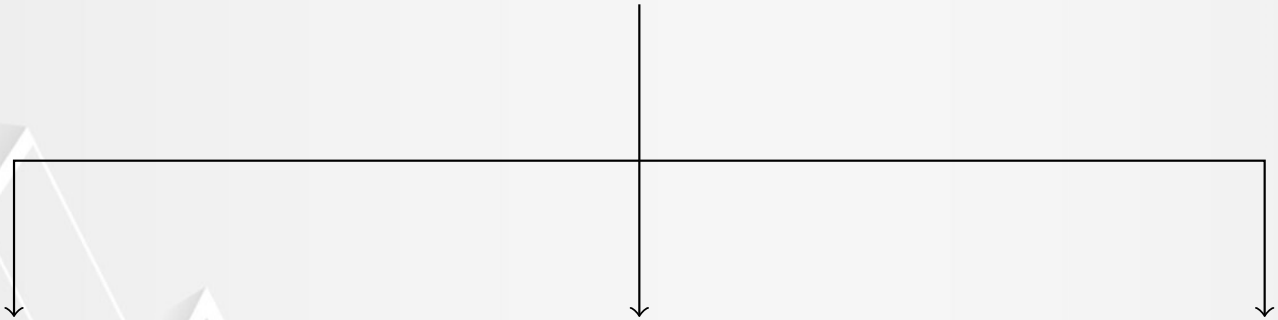
$$\begin{bmatrix} Q_{11} & Q_{12} & \cdots \\ Q_{21} & Q_{22} & \cdots \\ \vdots & \vdots & \vdots \end{bmatrix}$$

Encoder Output

$$X'$$

$$\begin{bmatrix} x'_{11} & x'_{12} & \cdots & \cdots \\ x'_{21} & x'_{22} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

(seq, 512)



$$Q$$

$$\begin{bmatrix} Q_{11} & Q_{12} & \cdots \\ Q_{21} & Q_{22} & \cdots \\ \vdots & \vdots & \vdots \end{bmatrix}$$

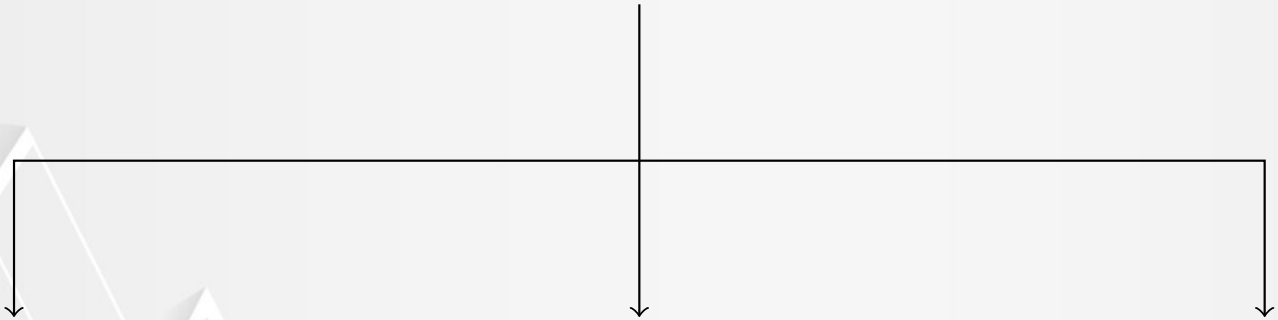
$$K$$

$$\begin{bmatrix} K_{11} & K_{12} & \cdots \\ K_{21} & K_{22} & \cdots \\ \vdots & \vdots & \vdots \end{bmatrix}$$

Encoder Output

$$X' = \begin{bmatrix} x'_{11} & x'_{12} & \cdots & \cdots \\ x'_{21} & x'_{22} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

(seq, 512)



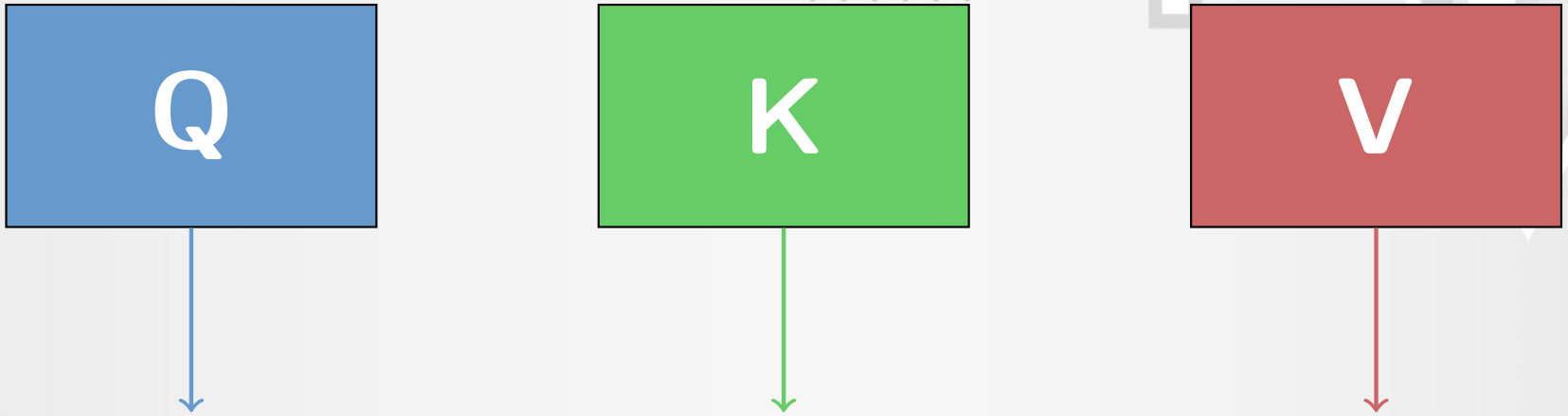
$$Q = \begin{bmatrix} Q_{11} & Q_{12} & \cdots \\ Q_{21} & Q_{22} & \cdots \\ \vdots & \vdots & \vdots \end{bmatrix} \quad K = \begin{bmatrix} K_{11} & K_{12} & \cdots \\ K_{21} & K_{22} & \cdots \\ \vdots & \vdots & \vdots \end{bmatrix} \quad V = \begin{bmatrix} V_{11} & V_{12} & \cdots \\ V_{21} & V_{22} & \cdots \\ \vdots & \vdots & \vdots \end{bmatrix}$$

Q, K, and V

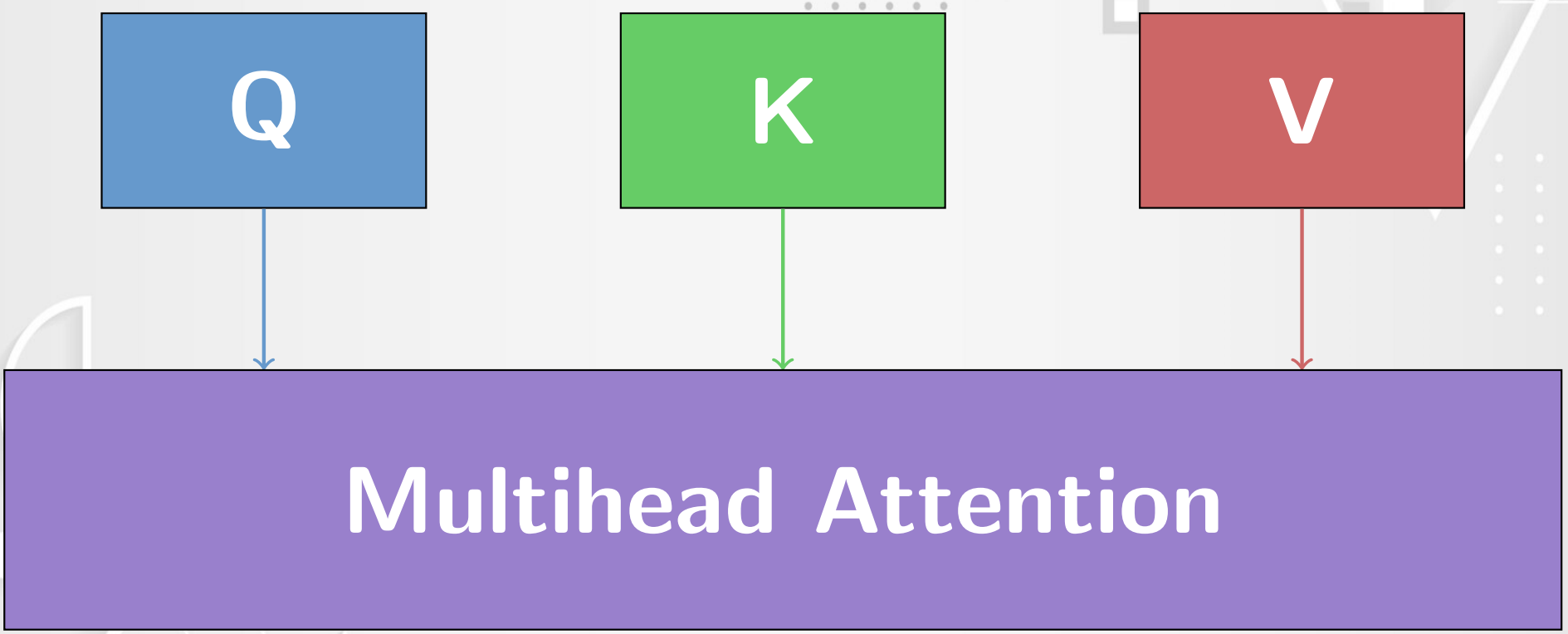
Q, K, and V

Q, K, and V

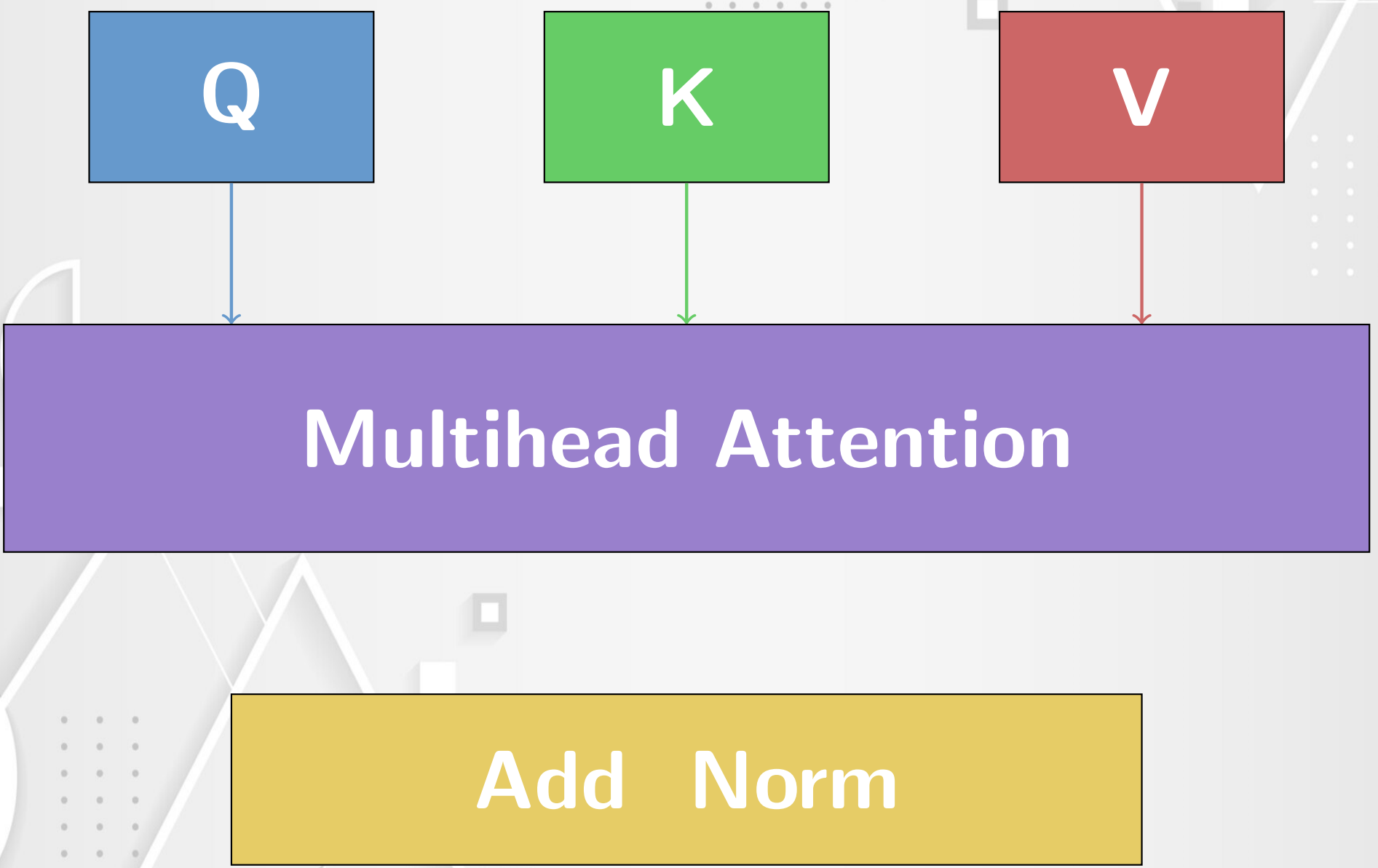
Q, K, and V



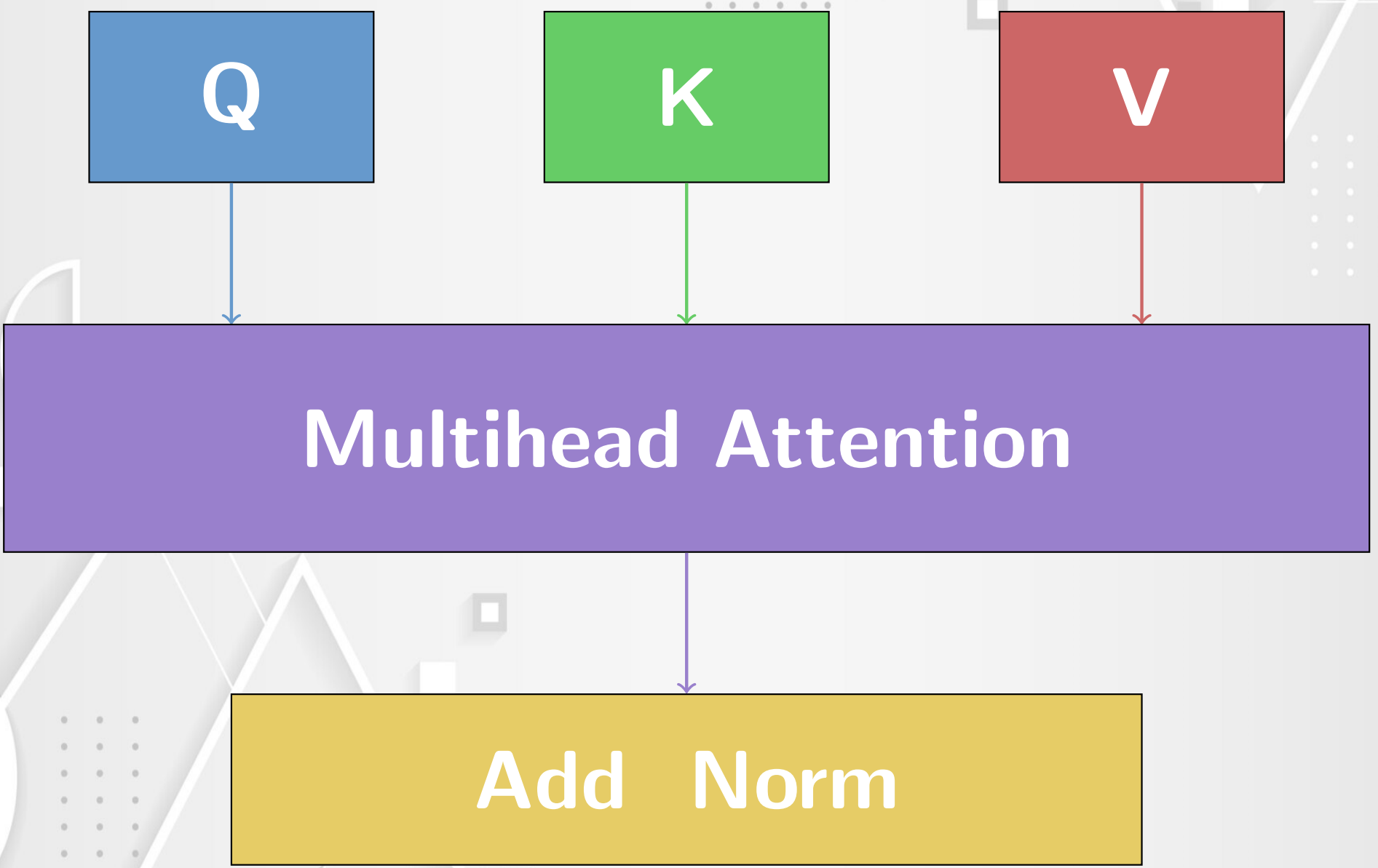
Q, K, and V



Q, K, and V



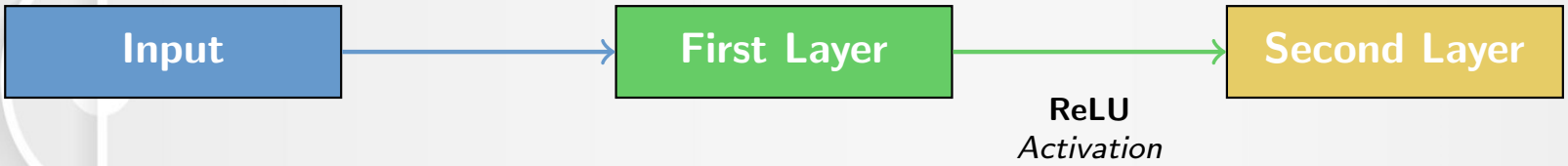
Q, K, and V



Encoder Layer - Feed Forward Network

Feed Forward Processing

Feed Forward Processing



Mathematical Formulation for Feedforward Network

- **Given the output from the attention mechanism Z (for each token):**

Mathematical Formulation for Feedforward Network

- **Given the output from the attention mechanism Z (for each token):**
- **First Linear Transformation:**

$$A_1 = \text{ReLU}(ZW_1 + b_1)$$

Mathematical Formulation for Feedforward Network

- **Given the output from the attention mechanism Z (for each token):**
- **First Linear Transformation:**

$$A_1 = \text{ReLU}(ZW_1 + b_1)$$

- **Second Linear Transformation:**

$$A_2 = A_1 W_2 + b_2$$

Mathematical Formulation for Feedforward Network

- **Given the output from the attention mechanism Z (for each token):**
- **First Linear Transformation:**

$$A_1 = \text{ReLU}(ZW_1 + b_1)$$

- **Second Linear Transformation:**

$$A_2 = A_1 W_2 + b_2$$

- **Add Residual Connection:**

$$Z_{\text{out}} = A_2 + Z$$

Mathematical Formulation for Feedforward Network

- **Given the output from the attention mechanism Z (for each token):**
- **First Linear Transformation:**

$$A_1 = \text{ReLU}(ZW_1 + b_1)$$

- **Second Linear Transformation:**

$$A_2 = A_1 W_2 + b_2$$

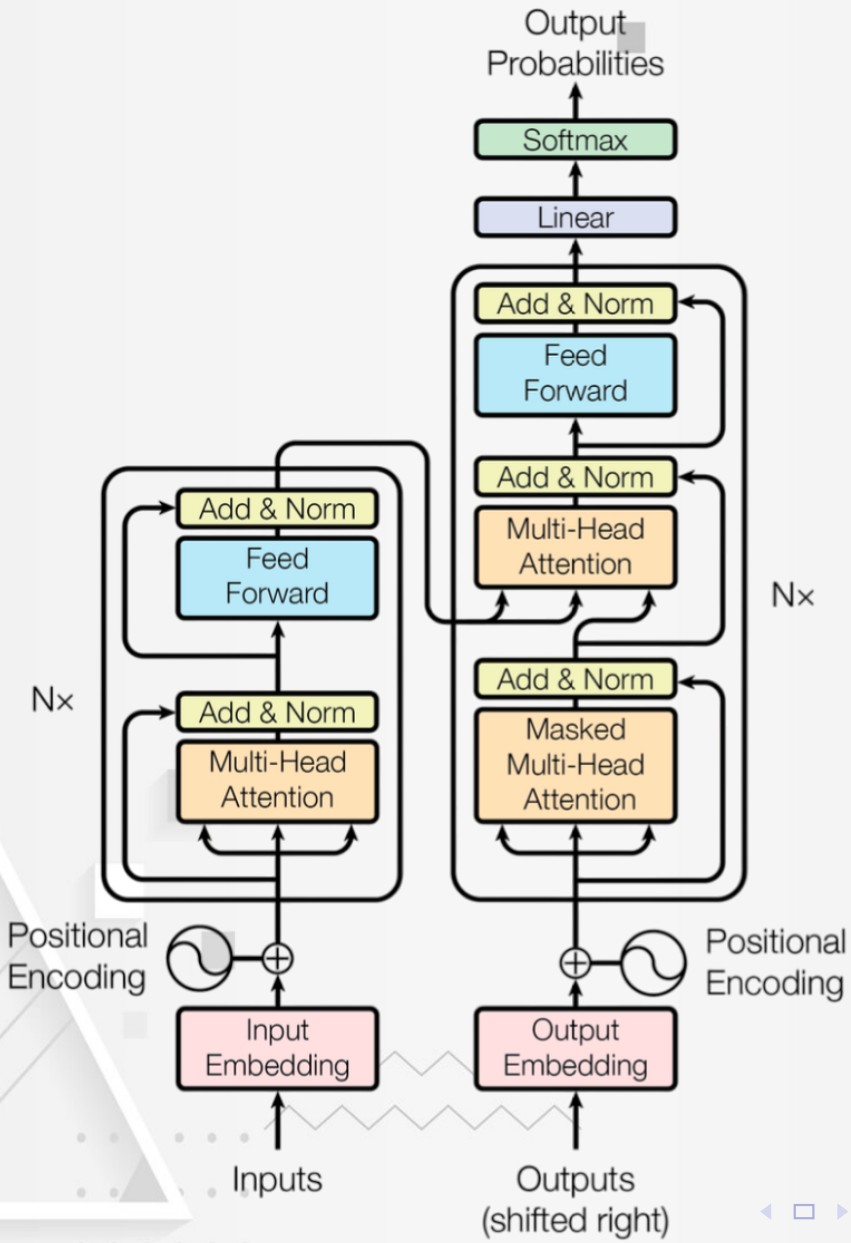
- **Add Residual Connection:**

$$Z_{\text{out}} = A_2 + Z$$

- **Layer Normalization:**

$$\text{Norm}(Z_{\text{out}})$$

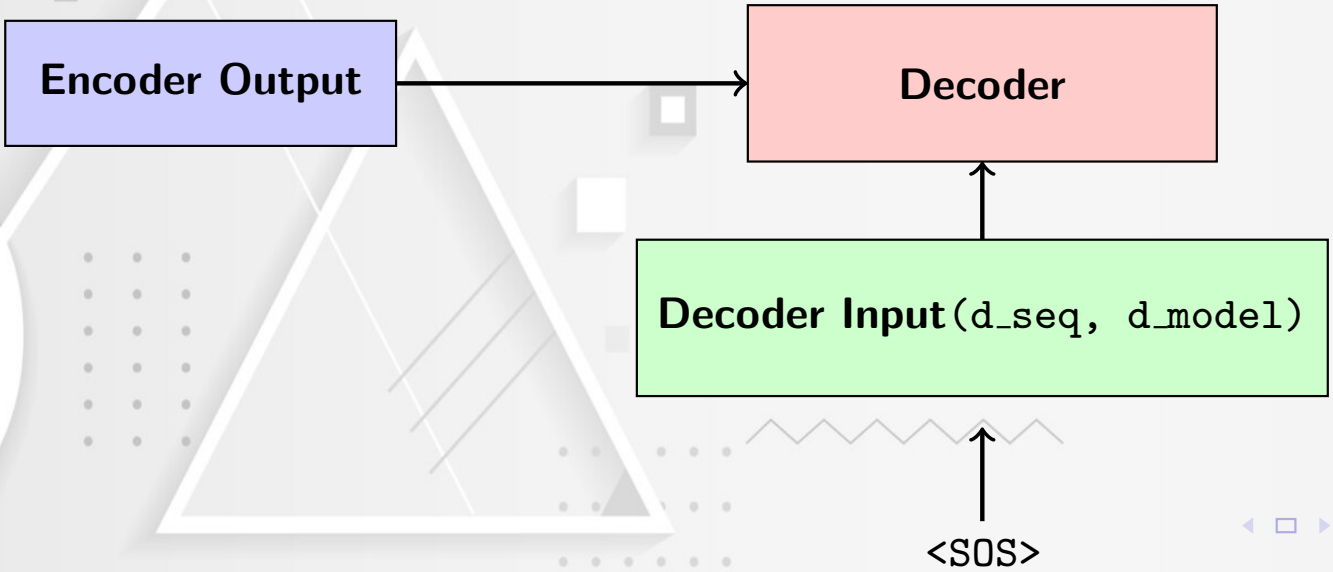
The Transformer



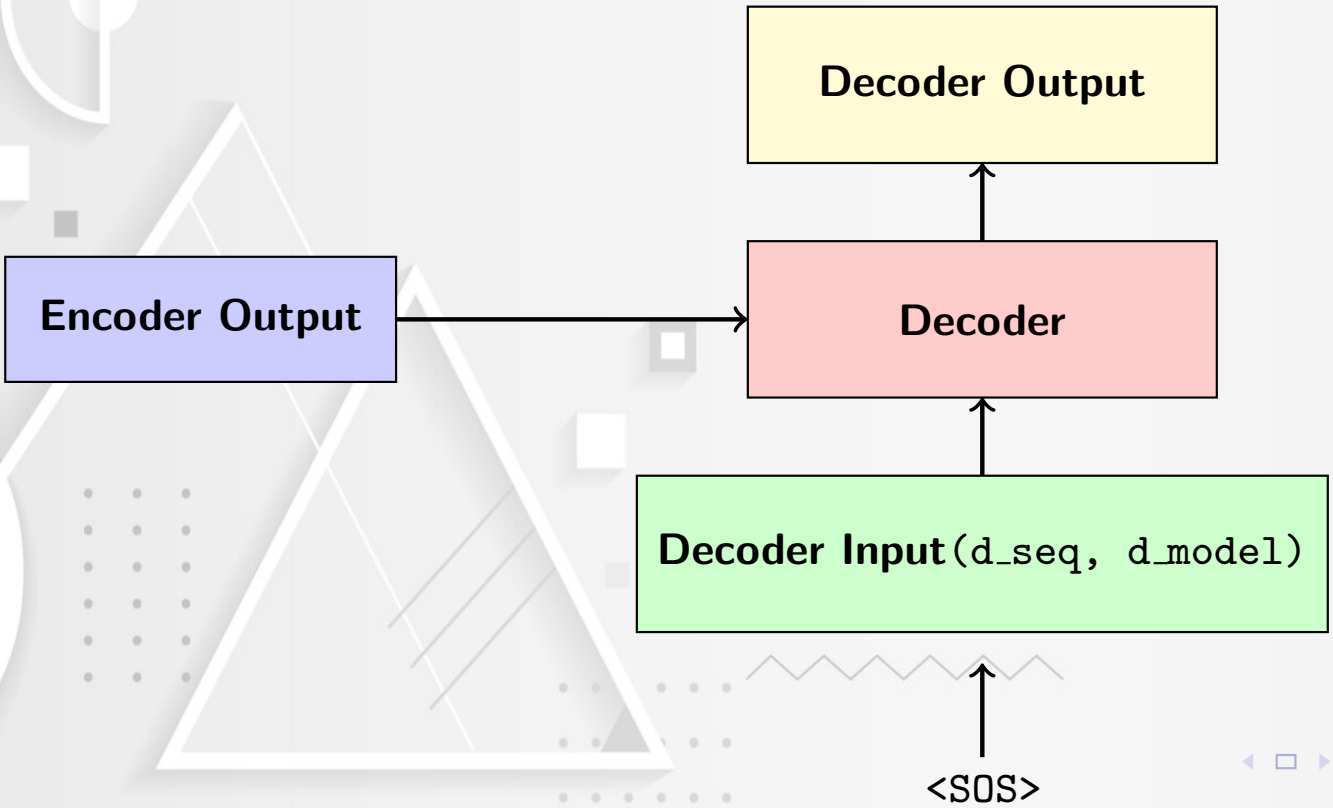
Masked Multi-Head Attention

Visualizing Masked Multihead Attention

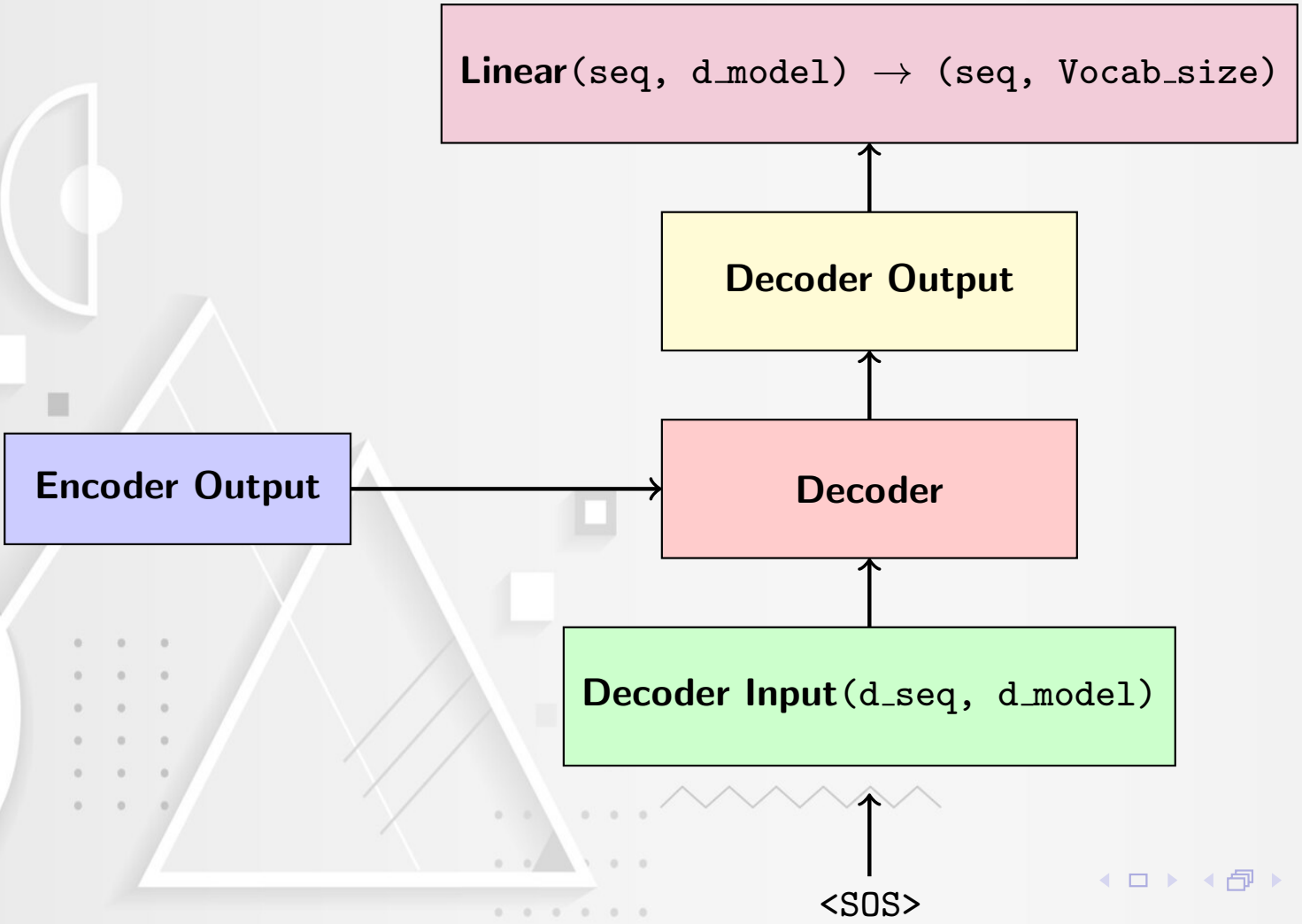
Total Flow



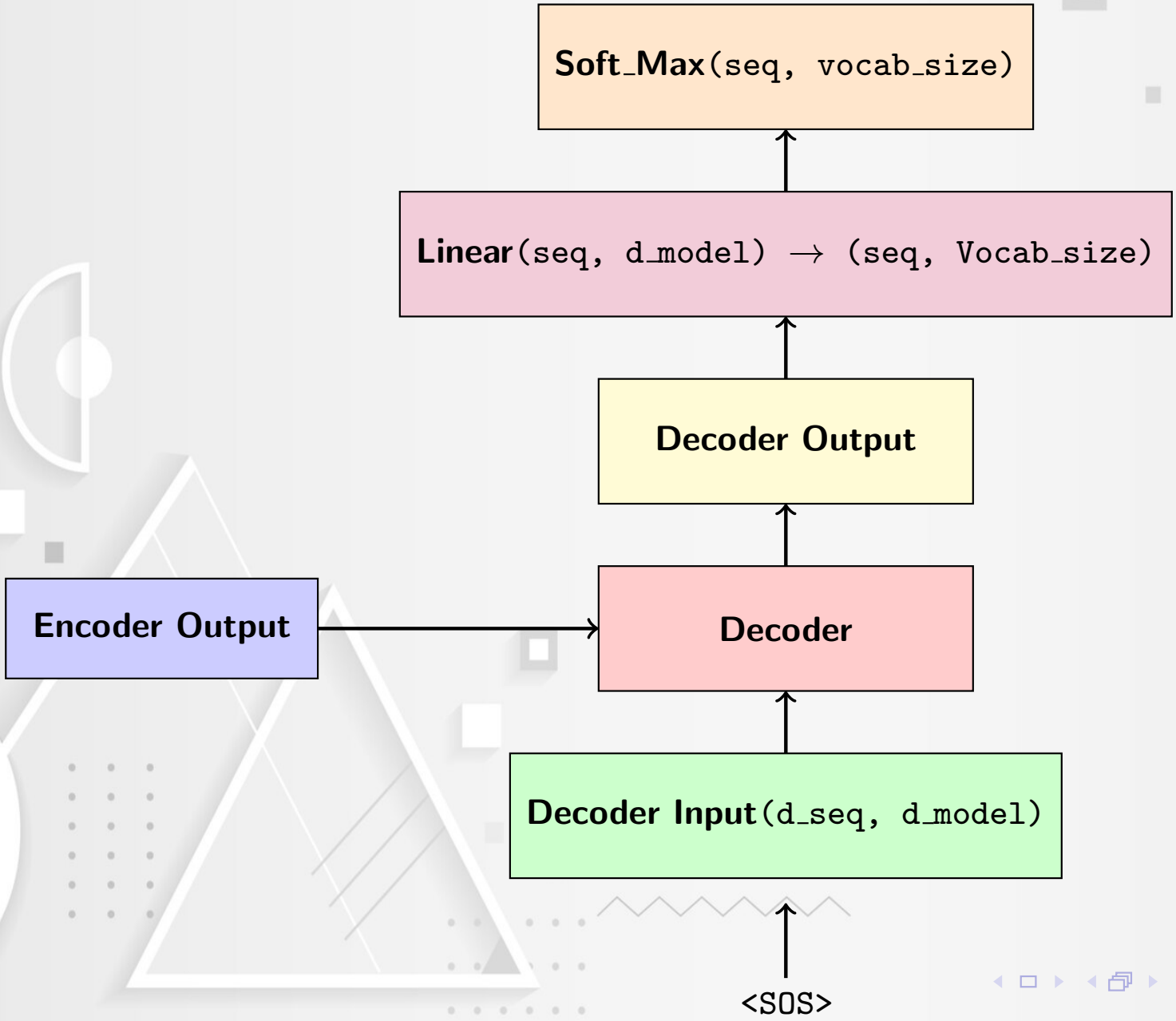
Total Flow


Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>

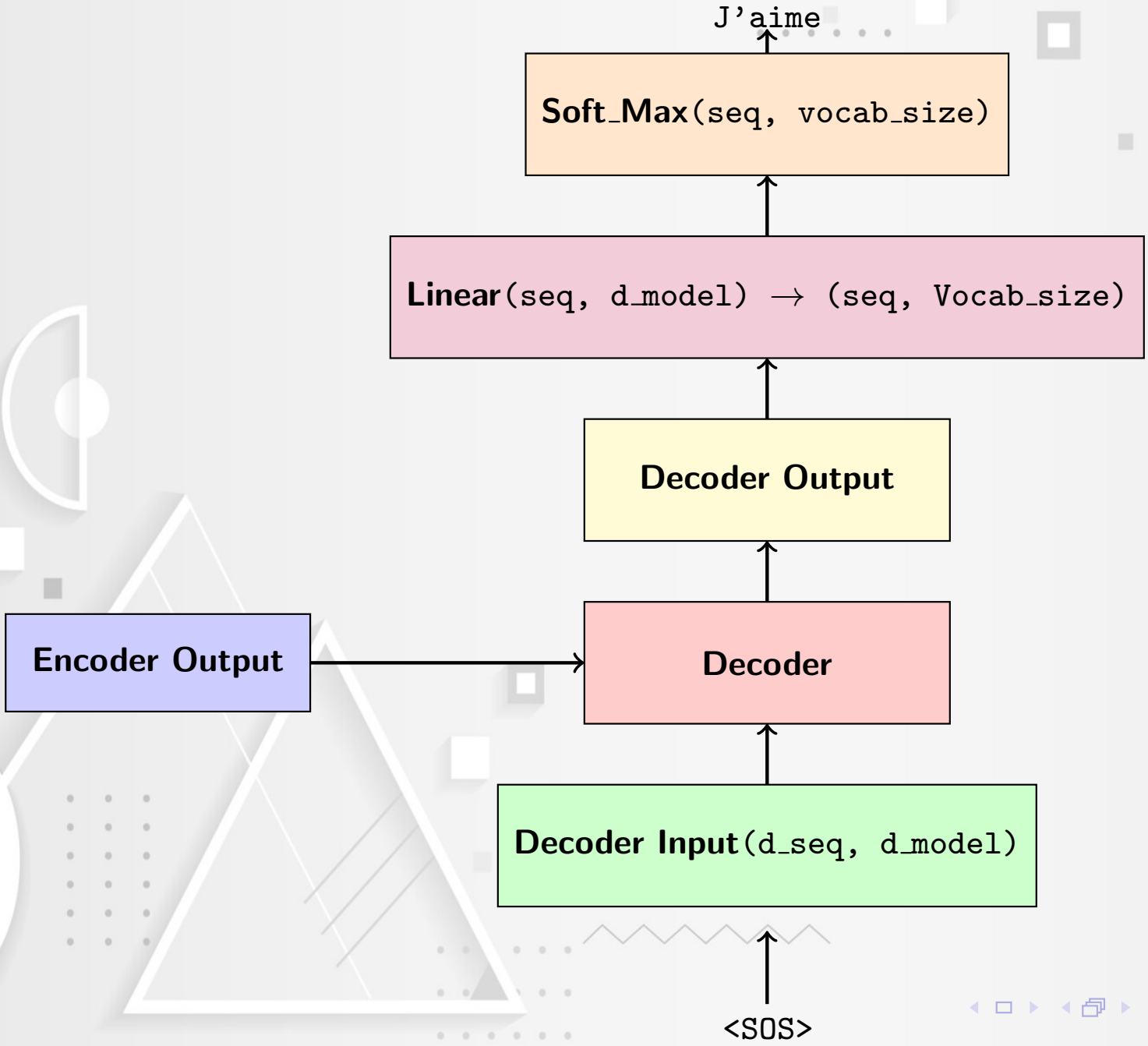
Total Flow


Linear(seq, d_model) → (seq, Vocab_size)
Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>

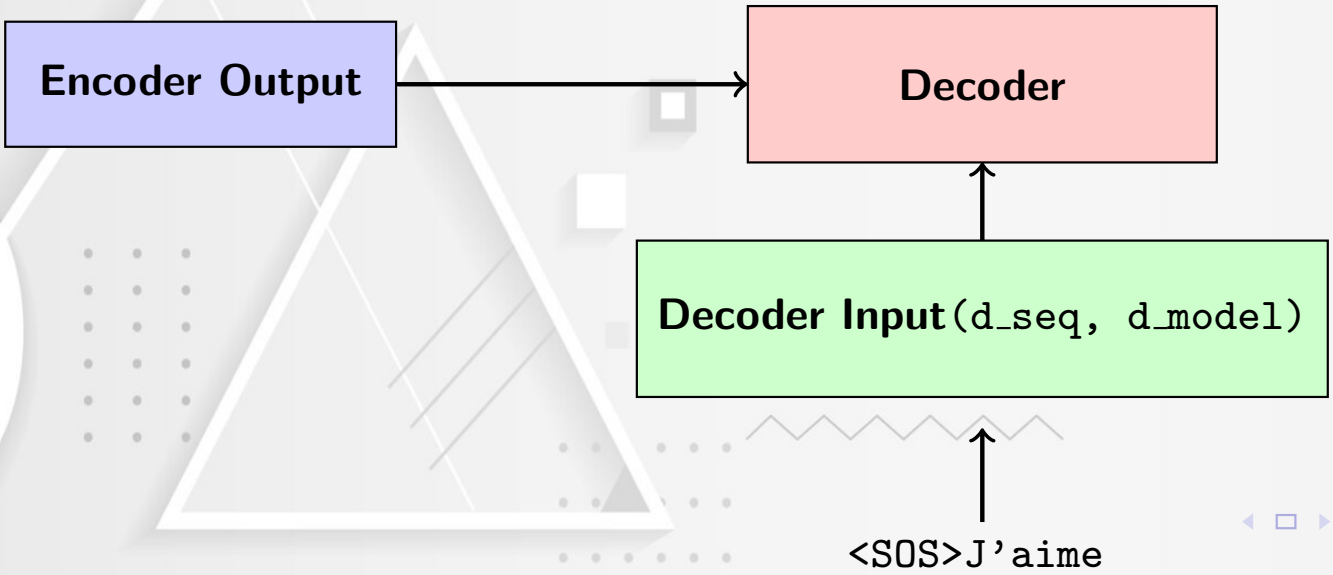
Total Flow


Soft_Max(seq, vocab_size)
Linear(seq, d_model) → (seq, Vocab_size)
Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>

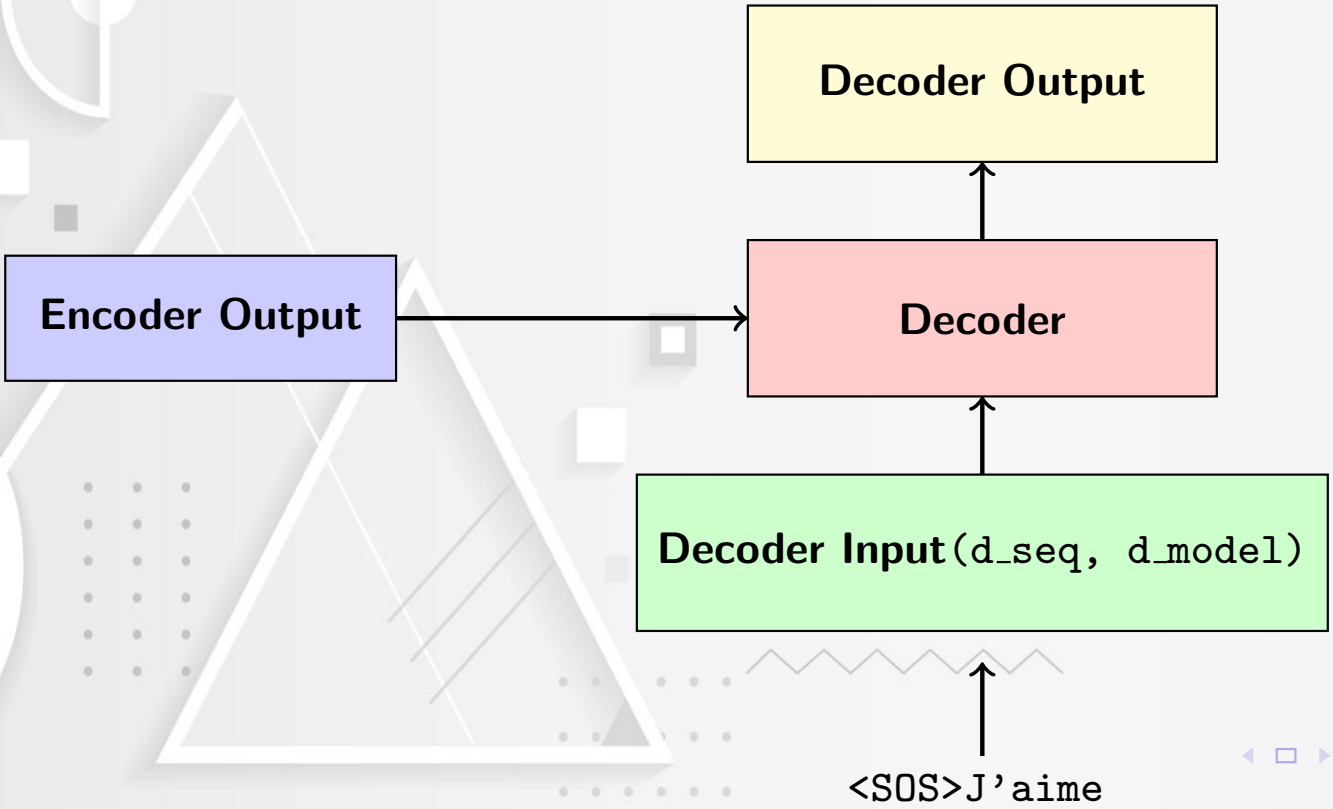
Total Flow


J'aime
Soft_Max(seq, vocab_size)
Linear(seq, d_model) → (seq, Vocab_size)
Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>

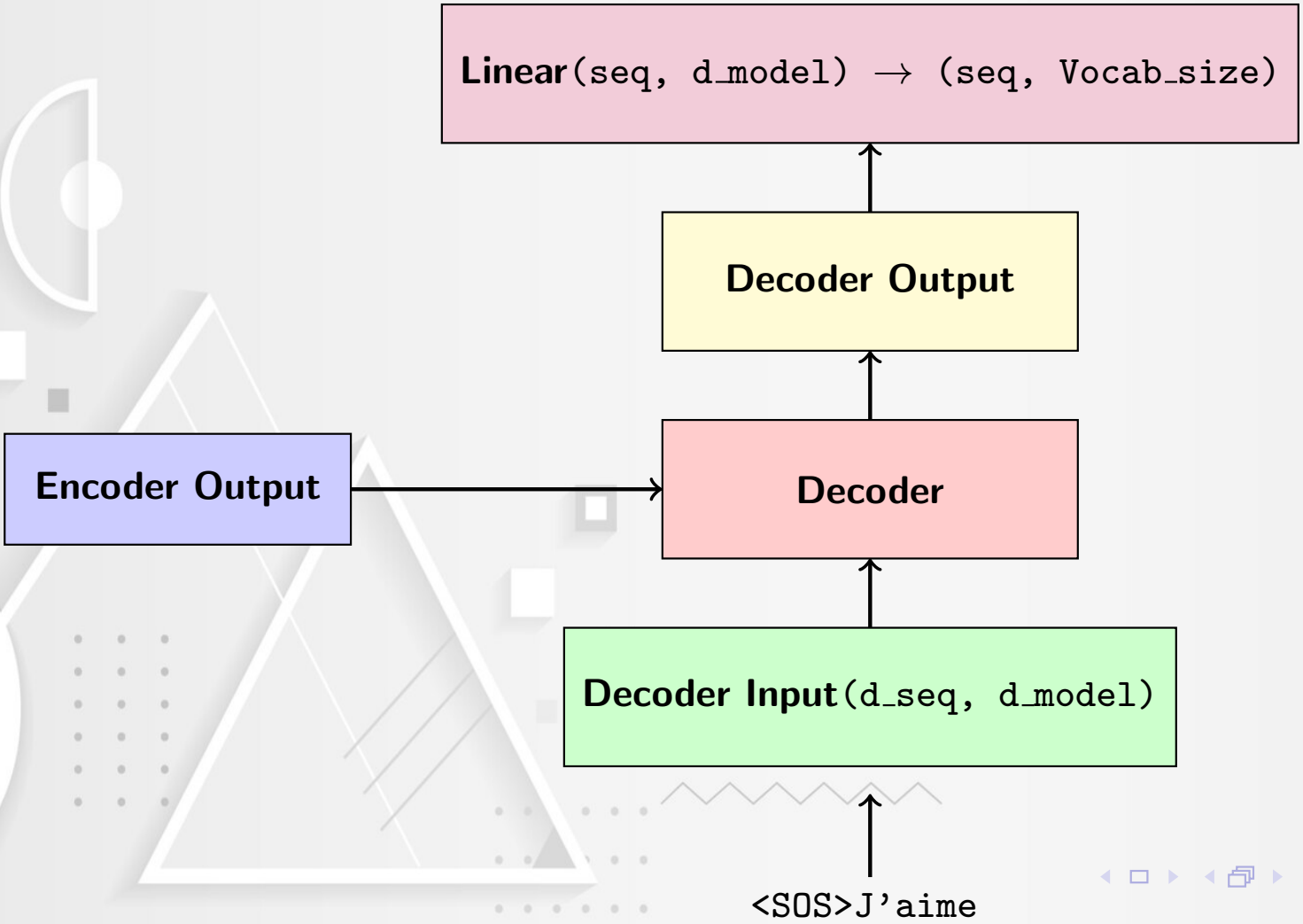
Total Flow


Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime

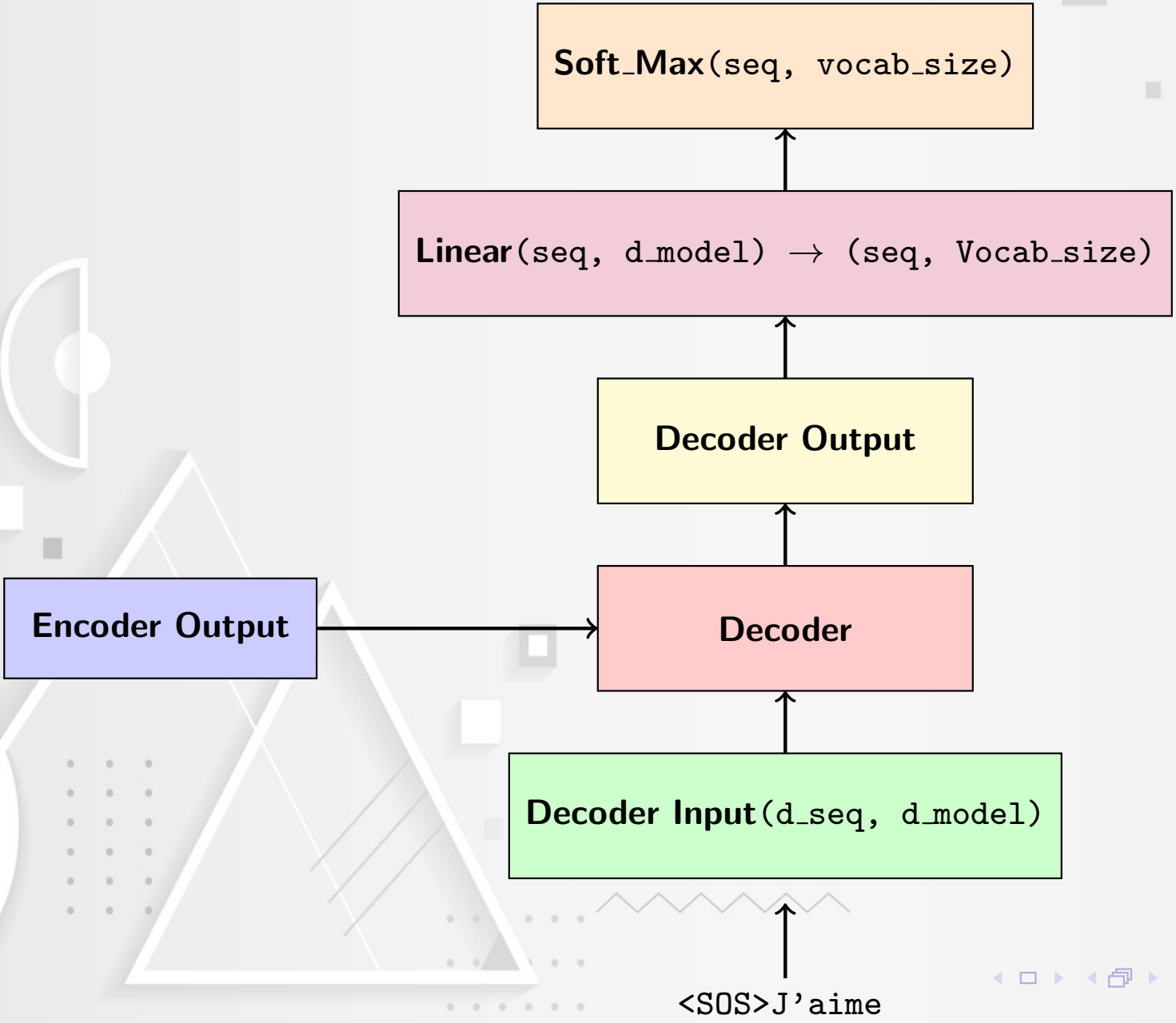
Total Flow


Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime

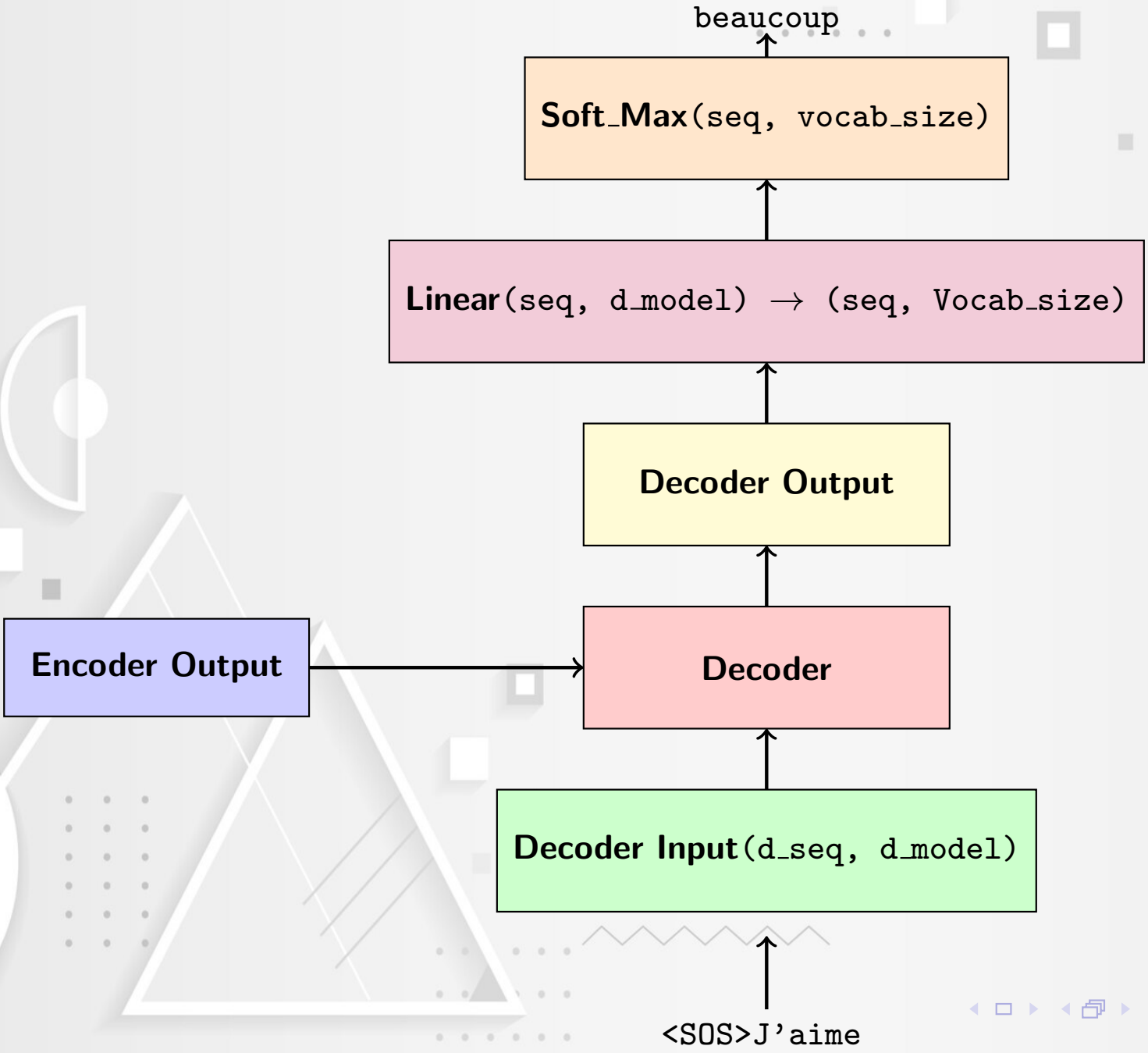
Total Flow


Linear(seq, d_model) → (seq, Vocab_size)
Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime

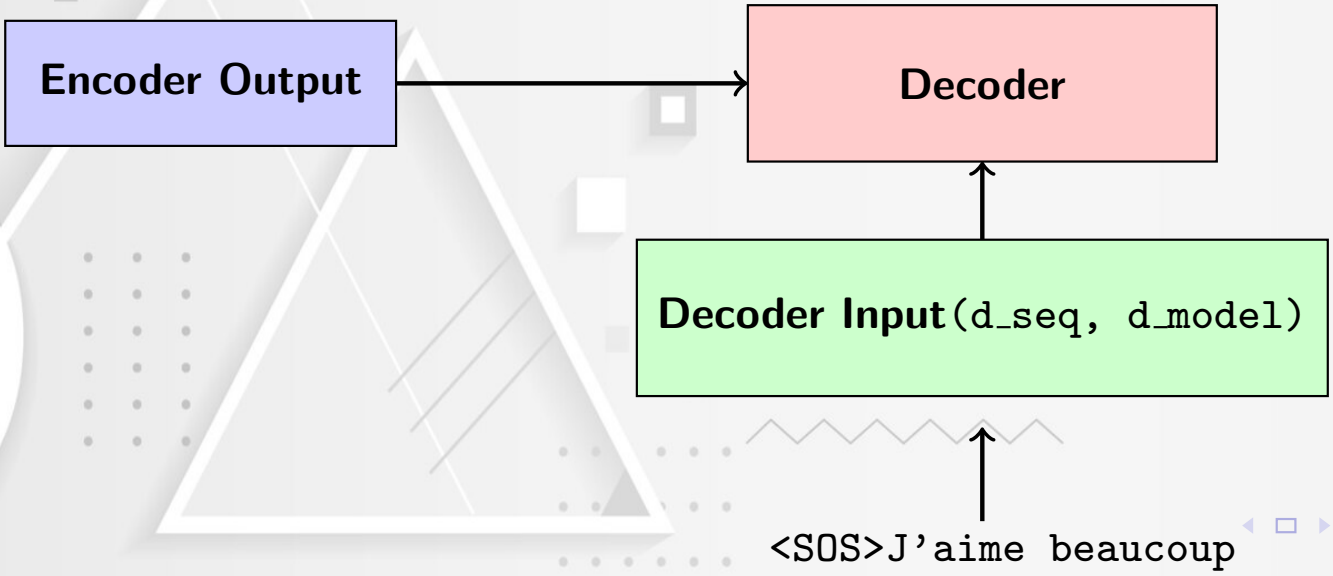
Total Flow


Soft_Max(seq, vocab_size)
Linear(seq, d_model) → (seq, Vocab_size)
Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime

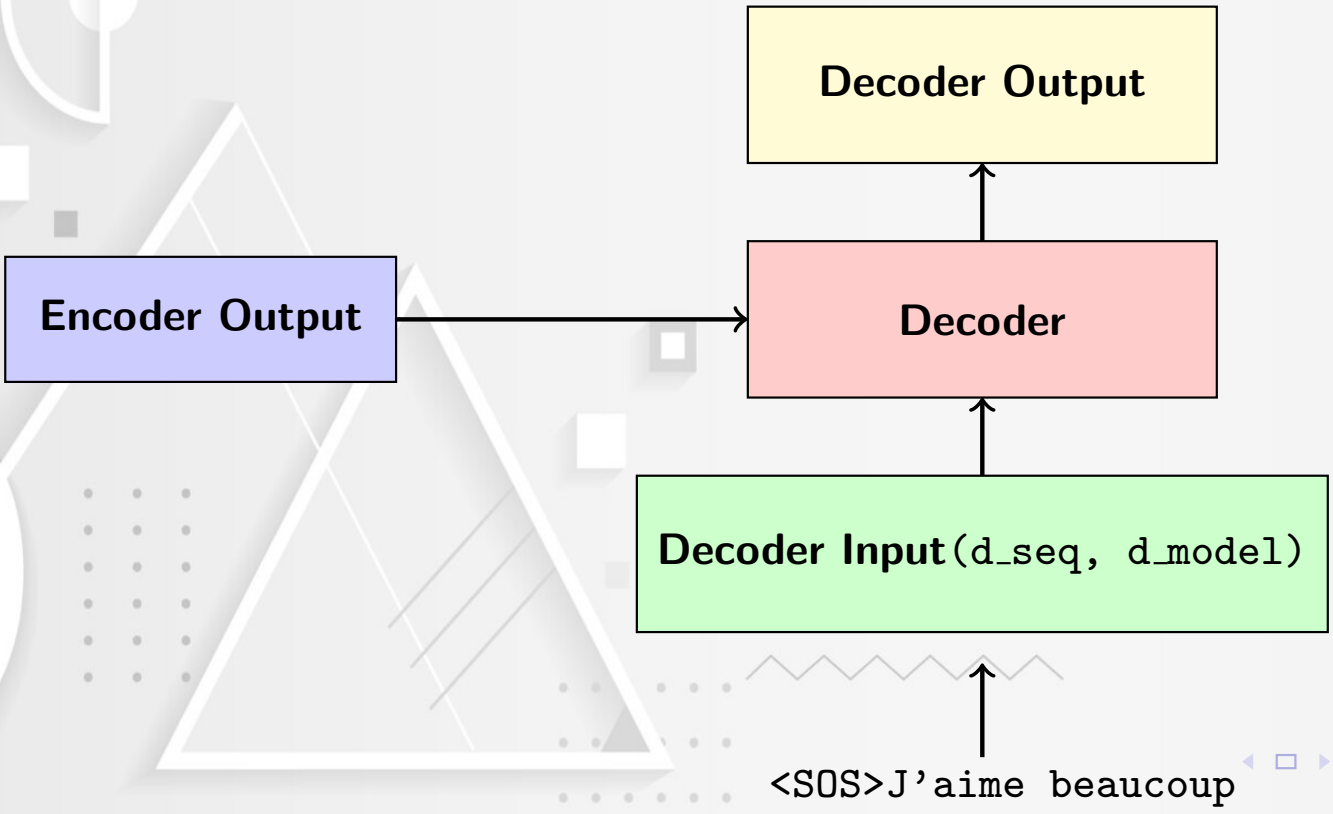
Total Flow


beaucoup
Soft_Max(seq, vocab_size)
Linear(seq, d_model) → (seq, Vocab_size)
Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime

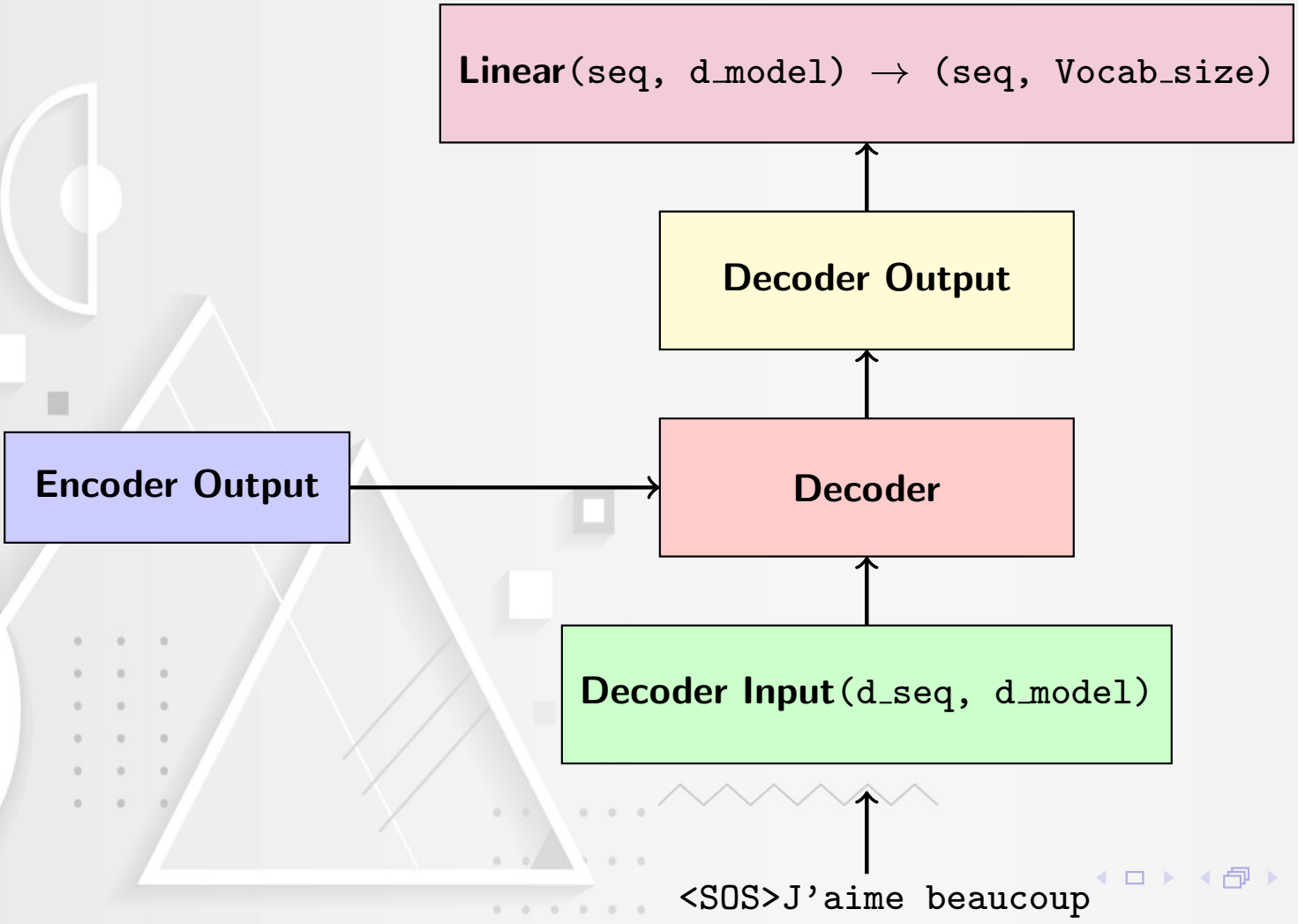
Total Flow


Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime beaucoup

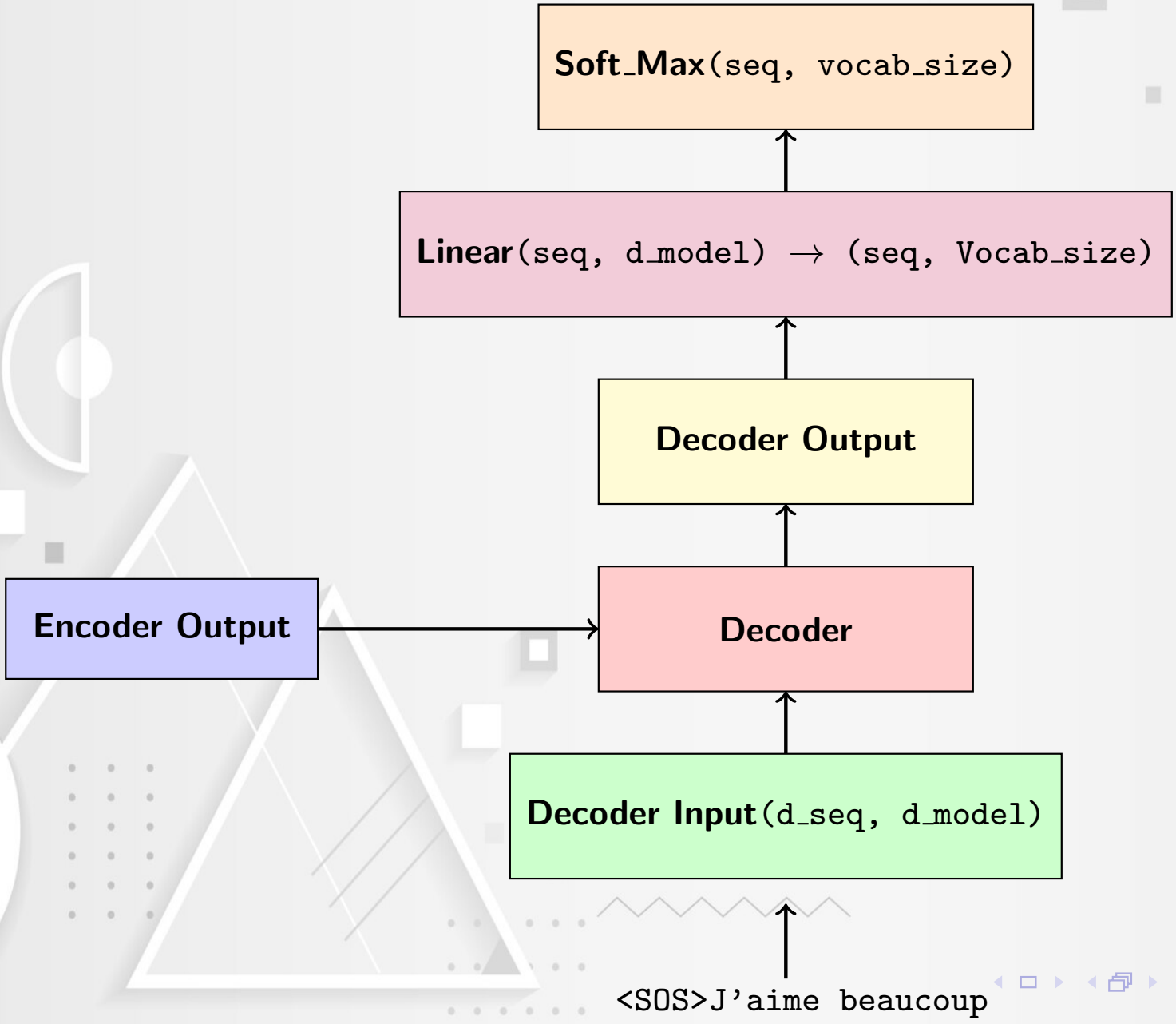
Total Flow


Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime beaucoup

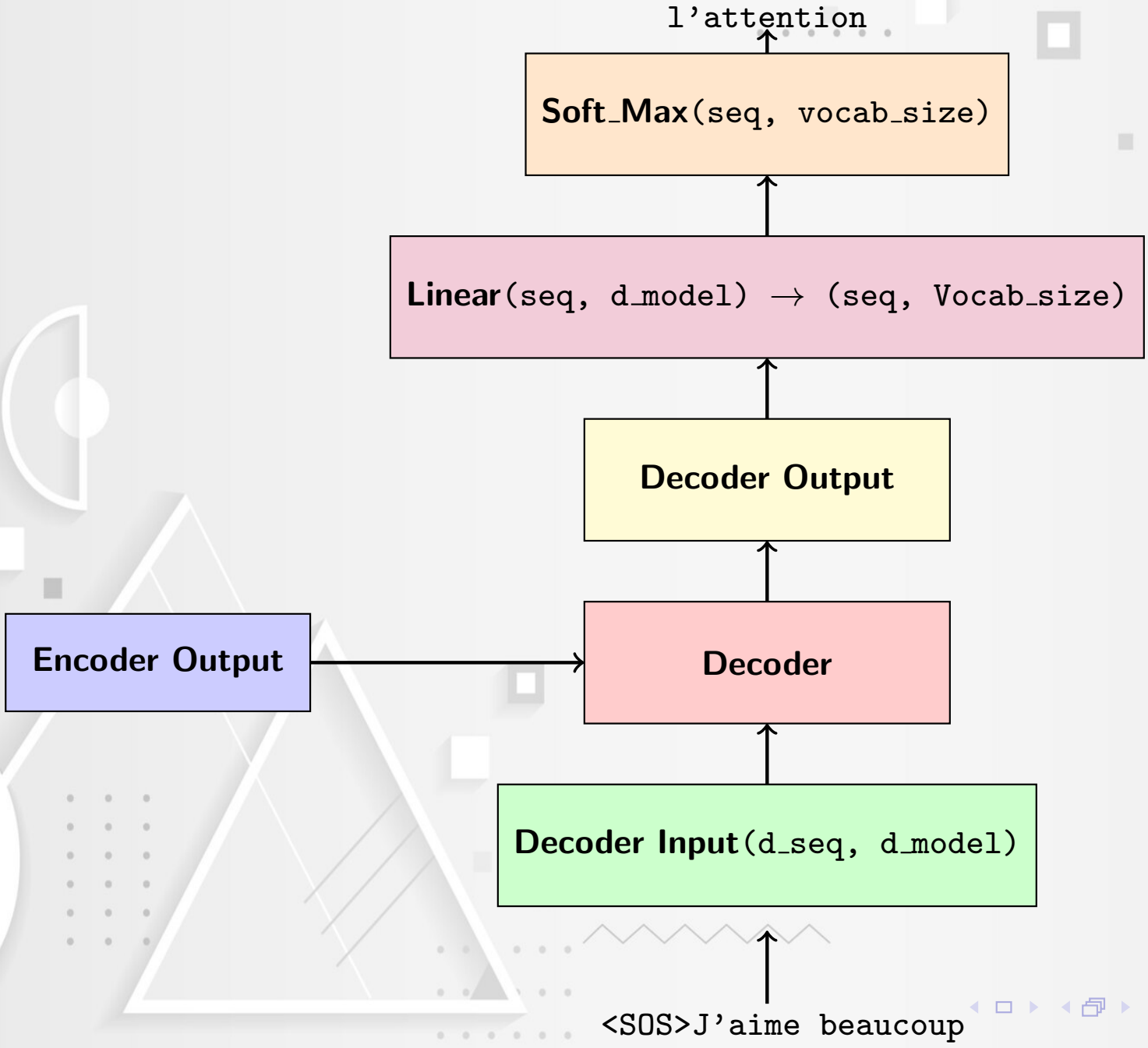
Total Flow


Linear(seq, d_model) → (seq, Vocab_size)
Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime beaucoup

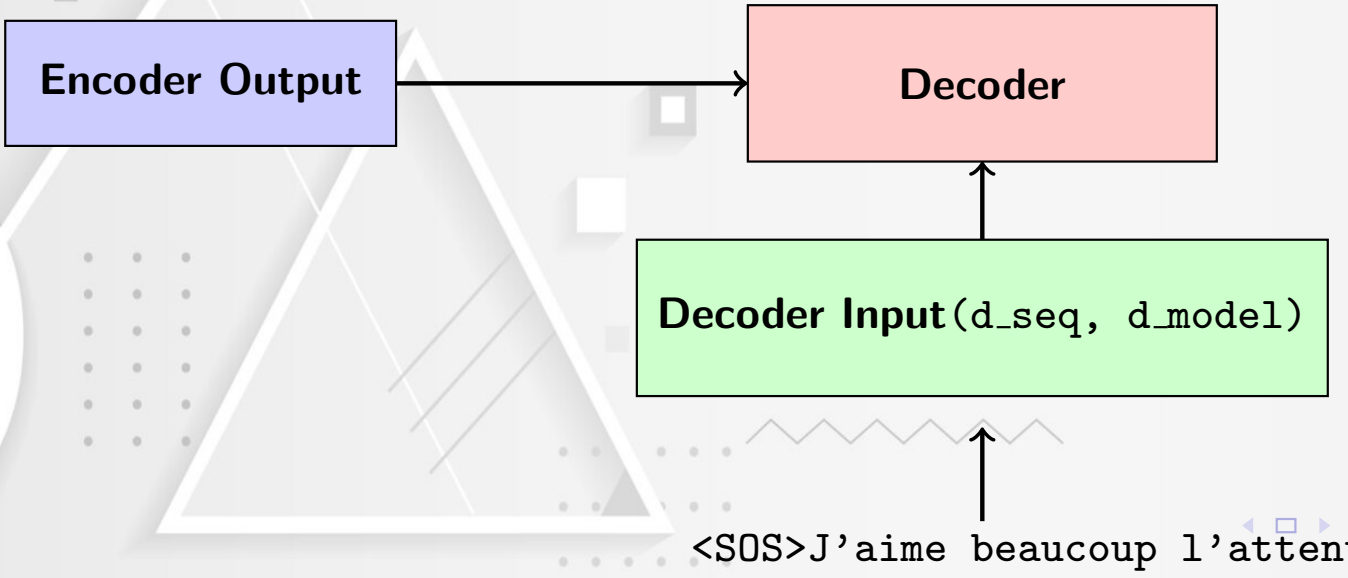
Total Flow


Soft_Max(seq, vocab_size)
Linear(seq, d_model) → (seq, Vocab_size)
Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime beaucoup

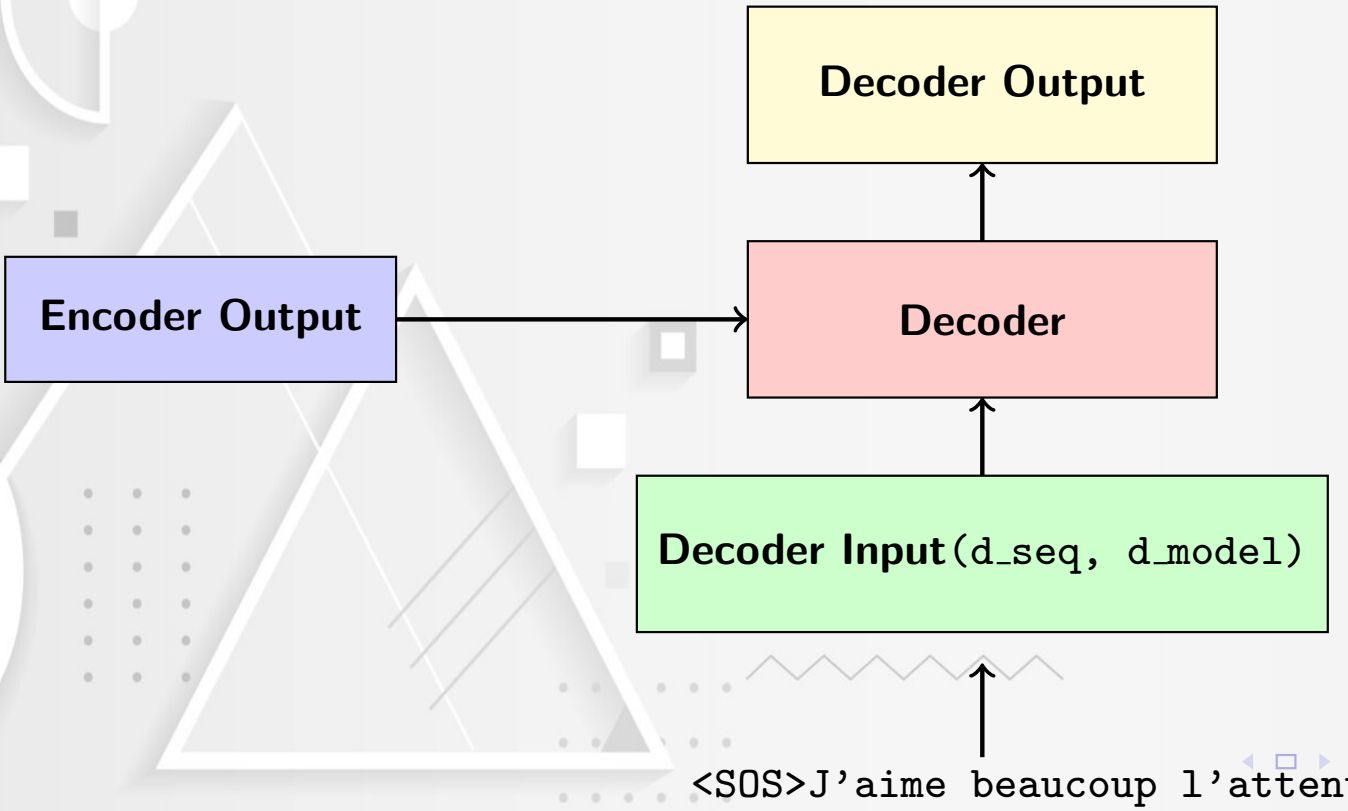
Total Flow


l'attention
Soft_Max(seq, vocab_size)
Linear(seq, d_model) → (seq, Vocab_size)
Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime beaucoup

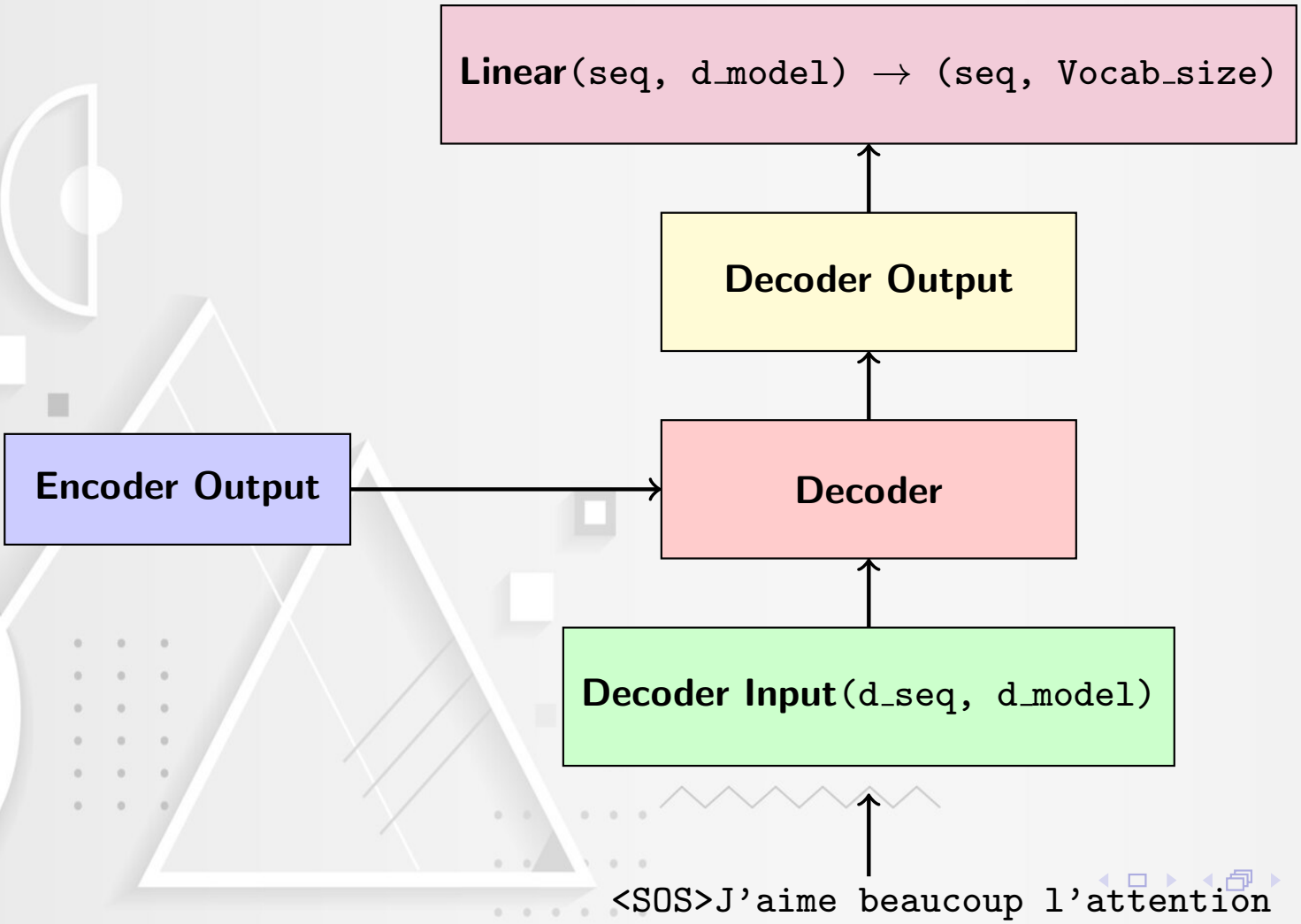
Total Flow


Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime beaucoup l'attention

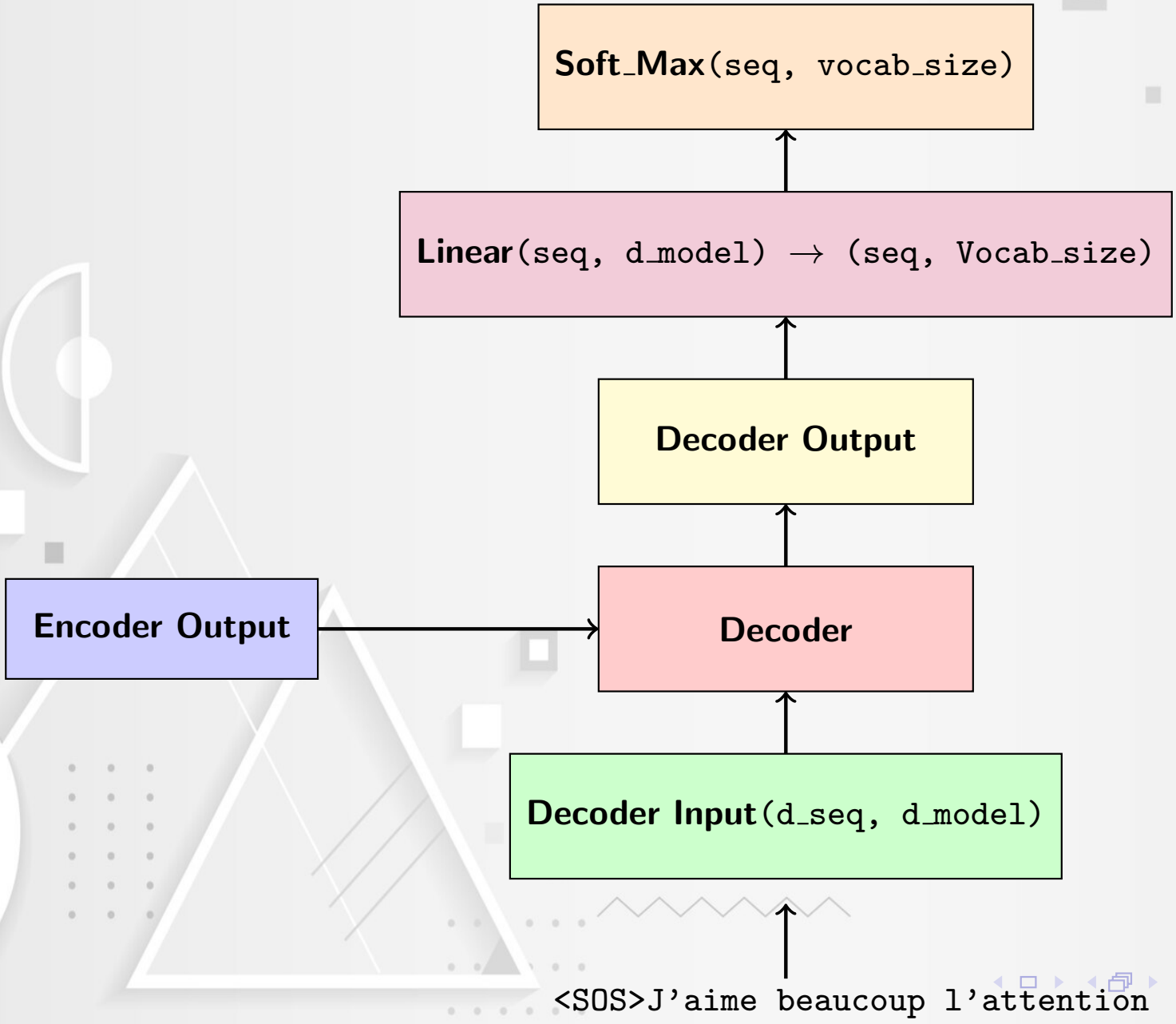
Total Flow


Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime beaucoup l'attention

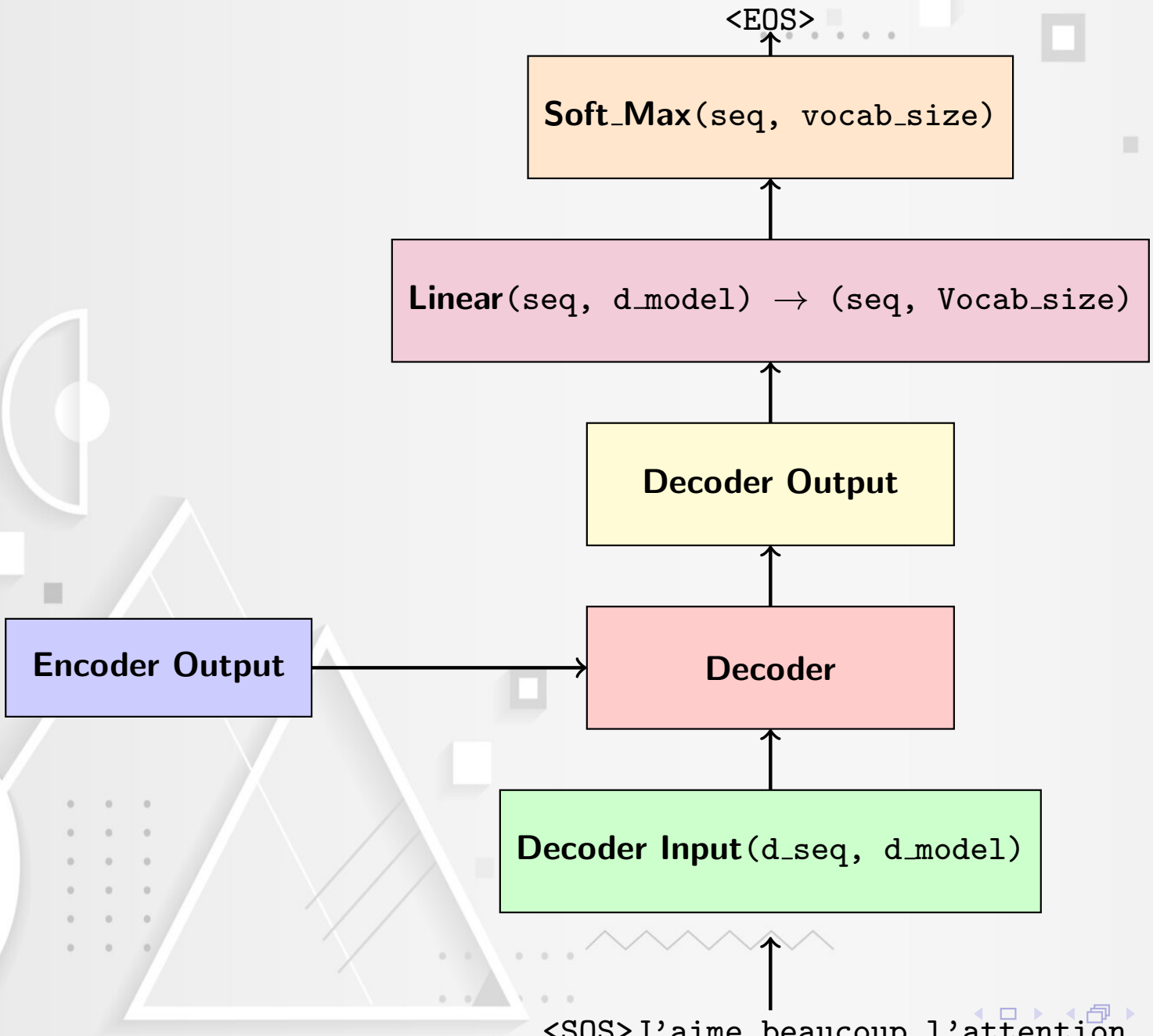
Total Flow


Linear(seq, d_model) → (seq, Vocab_size)
Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime beaucoup l'attention

Total Flow


Soft_Max(seq, vocab_size)
Linear(seq, d_model) → (seq, Vocab_size)
Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime beaucoup l'attention

Total Flow


<EOS>
Soft_Max(seq, vocab_size)
Linear(seq, d_model) → (seq, Vocab_size)
Decoder Output
Encoder Output
Decoder
Decoder Input(d_seq, d_model)
<SOS>J'aime beaucoup l'attention

Machine Translation Results (6.1)

- Transformer (big) outperforms all prior models on English-to-German with a BLEU score of **28.4**.

Machine Translation Results (6.1)

- Transformer (big) outperforms all prior models on English-to-German with a BLEU score of **28.4**.
- On English-to-French, Transformer achieves **41.0** BLEU, with **1/4 the training cost**.

Machine Translation Results (6.1)

- Transformer (big) outperforms all prior models on English-to-German with a BLEU score of **28.4**.
- On English-to-French, Transformer achieves **41.0** BLEU, with **1/4 the training cost**.
- Base models averaged the last 5 checkpoints, and beam search was used with beam size $= 4$.

Machine Translation Results (6.1)

- Transformer (big) outperforms all prior models on English-to-German with a BLEU score of **28.4**.
- On English-to-French, Transformer achieves **41.0** BLEU, with **1/4 the training cost**.
- Base models averaged the last 5 checkpoints, and beam search was used with beam size $= 4$.
- Training cost is compared with other architectures, and estimated using TFLOPS.

Machine Translation Results (6.1)

- Transformer (big) outperforms all prior models on English-to-German with a BLEU score of **28.4**.
- On English-to-French, Transformer achieves **41.0** BLEU, with **1/4 the training cost**.
- Base models averaged the last 5 checkpoints, and beam search was used with beam size $= 4$.
- Training cost is compared with other architectures, and estimated using TFLOPS.

Model Variations (6.2)

- **Experiment with Transformer Components:** Varying configurations and measuring BLEU score on English-to-German (newstest2013).

Model Variations (6.2)

- **Experiment with Transformer Components:** Varying configurations and measuring BLEU score on English-to-German (newstest2013).
- **Key Findings:**
  - **Attention Heads:** Single-head attention is 0.9 BLEU worse (Row A).

Model Variations (6.2)

- **Experiment with Transformer Components:** Varying configurations and measuring BLEU score on English-to-German (newstest2013).
- **Key Findings:**
  - **Attention Heads:** Single-head attention is 0.9 BLEU worse (Row A).
  - **Attention Key Size (dk):** Smaller dk reduces quality (Row B).

Model Variations (6.2)

- **Experiment with Transformer Components:** Varying configurations and measuring BLEU score on English-to-German (newstest2013).
- **Key Findings:**
  - **Attention Heads:** Single-head attention is 0.9 BLEU worse (Row A).
  - **Attention Key Size (dk):** Smaller dk reduces quality (Row B).
  - **Model Size:** Larger models outperform smaller ones (Rows C, D).

Model Variations (6.2)

- **Experiment with Transformer Components:** Varying configurations and measuring BLEU score on English-to-German (newstest2013).
- **Key Findings:**
  - **Attention Heads:** Single-head attention is 0.9 BLEU worse (Row A).
  - **Attention Key Size (dk):** Smaller dk reduces quality (Row B).
  - **Model Size:** Larger models outperform smaller ones (Rows C, D).
  - **Dropout:** Crucial to prevent overfitting.

Model Variations (6.2)

- **Experiment with Transformer Components:** Varying configurations and measuring BLEU score on English-to-German (newstest2013).
- **Key Findings:**
  - **Attention Heads:** Single-head attention is 0.9 BLEU worse (Row A).
  - **Attention Key Size (dk):** Smaller dk reduces quality (Row B).
  - **Model Size:** Larger models outperform smaller ones (Rows C, D).
  - **Dropout:** Crucial to prevent overfitting.
  - **Positional Encoding:** Learned embeddings perform similarly to sinusoidal (Row E).

Model Variations (6.2)

- **Experiment with Transformer Components:** Varying configurations and measuring BLEU score on English-to-German (newstest2013).
- **Key Findings:**
  - **Attention Heads:** Single-head attention is 0.9 BLEU worse (Row A).
  - **Attention Key Size (dk):** Smaller dk reduces quality (Row B).
  - **Model Size:** Larger models outperform smaller ones (Rows C, D).
  - **Dropout:** Crucial to prevent overfitting.
  - **Positional Encoding:** Learned embeddings perform similarly to sinusoidal (Row E).
- **Table 3:** Summary of model variations, BLEU scores, and perplexities.

Row	Config	BLEU	PPL
(A)	Attention heads	25.8	5.29
(B)	Attention key size (dk)	24.9	5.00
(C)	Bigger model	25.5	4.91
(D)	Dropout rate	25.8	5.16
(E)	Learned positional embedding	26.0	5.47

Thank You for Your Attention