02

03

04

05

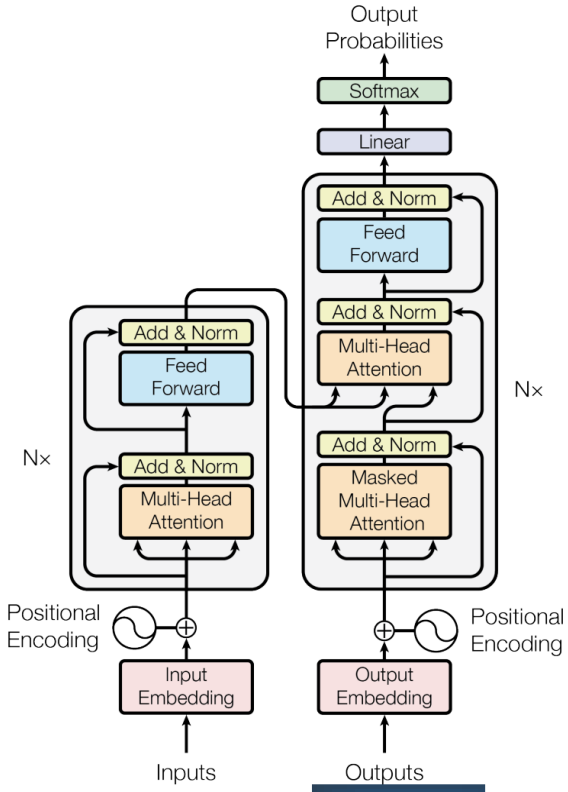
Chapitre 2 : Transformers

BERT

Encoder

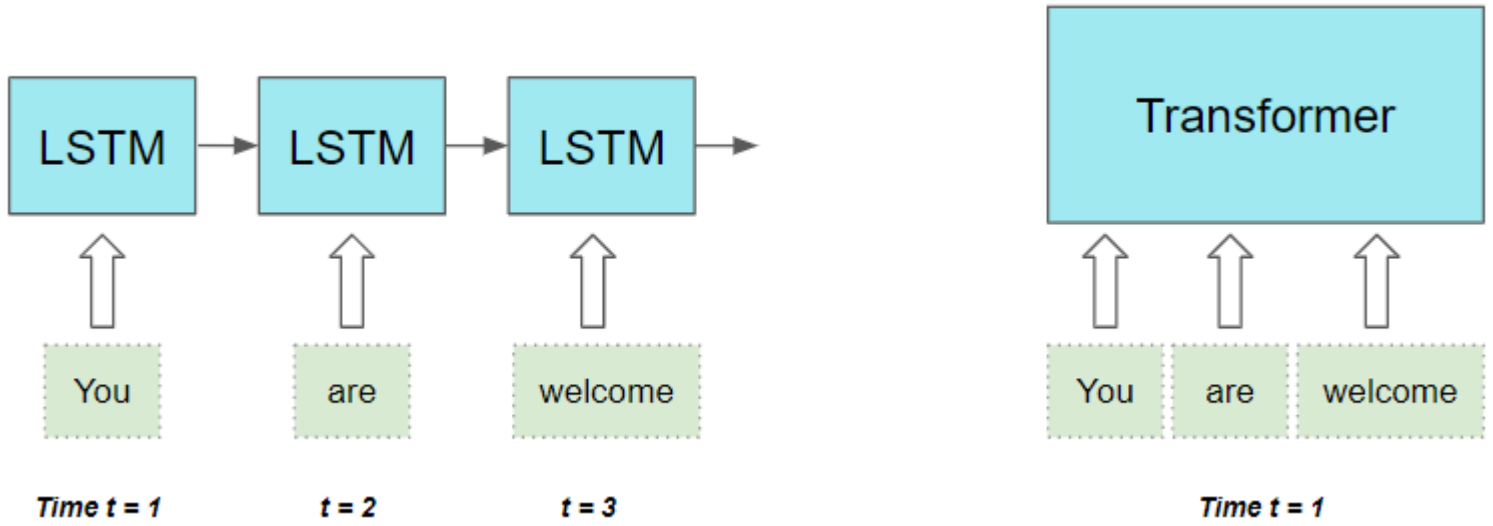
GPT

Decoder

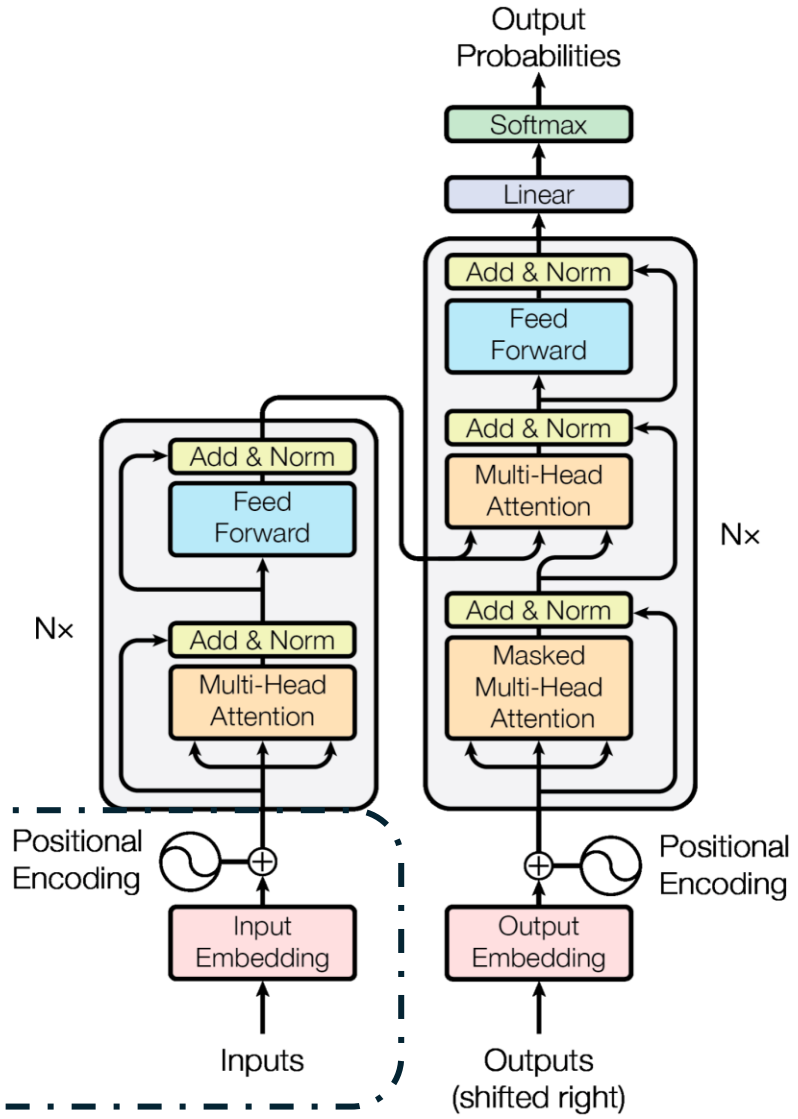


Transformers

En 2017, l'article **« Attention Is All You Need »** a introduit l'architecture Transformer, marquant sans doute la percée la plus importante du NLP depuis l'apparition des réseaux de neurones. Le Transformer a résolu les limitations fondamentales des RNN et des LSTM en traitant les séquences entières simultanément, plutôt que mot par mot de manière séquentielle..



Transformers



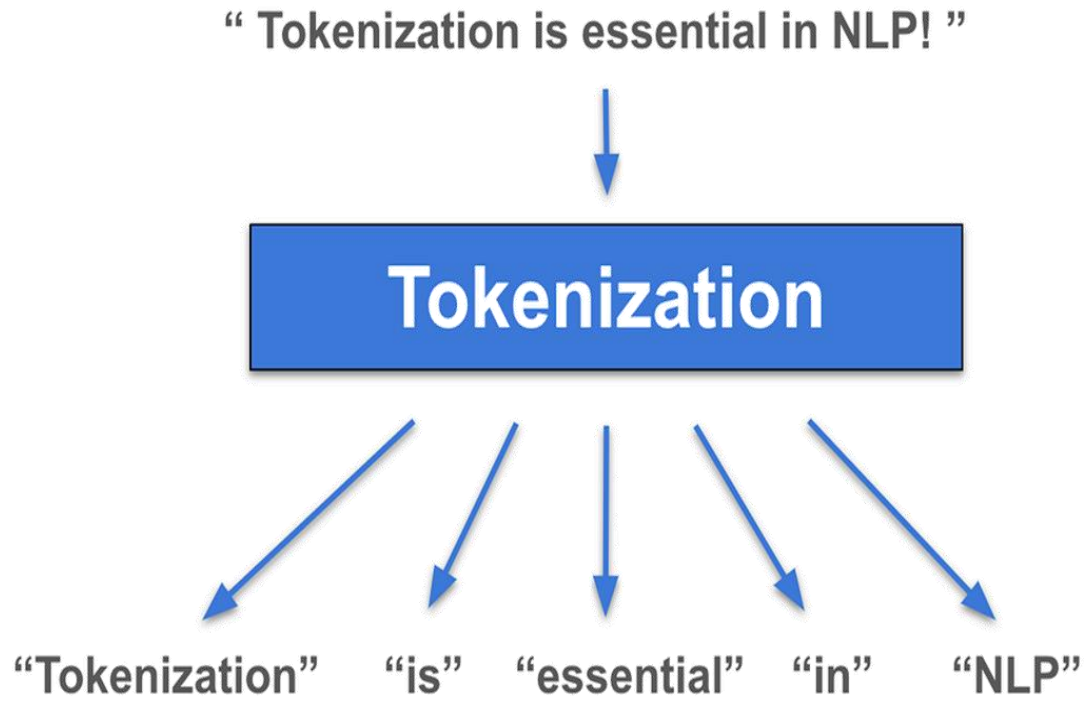
L'Embedding Layer avec Positional Encoding transforme la séquence d'entrée en une représentation vectorielle continue intégrant à la fois le sens des tokens et leur position dans la phrase.

Transformers

Embedding

1. Step 1: Tokenization
2. Step 2. Token Embedding
3. Step 3. Positional Encoding
4. Step 4. Final Embedding

La tokenisation est le processus qui consiste à découper des données complexes comme des paragraphes en unités simples appelées tokens.



1 Step 1: Tokenization

Voyons quelques exemples d'algorithmes de tokénisation :

1. Tokenizer basé sur les mots

Le premier type de tokenizer qui vient à l'esprit est celui basé sur les mots. Il est généralement très facile à utiliser et configurable avec seulement quelques règles. Il donne souvent des résultats décents.

Split on spaces

Let's	do	tokenization!

Split on punctuation

Let	's	do	tokenization	!

1 Step 1: Tokenization

Voyons quelques exemples d'algorithmes de tokénisation :

1. Tokenizer basé sur les mots

Le premier type de tokenizer qui vient à l'esprit est celui basé sur les mots. Il est généralement très facile à utiliser et configurable avec seulement quelques règles. Il donne souvent des résultats décents.

Inconvénients :

1. Le modèle ne comprend pas que *chien* et *chiens* sont liés, car chaque mot a son identifiant distinct.
2. Les mots dérivés comme *maison* et *maisonnette* ne sont pas perçus comme similaires — perte d'information sémantique.
3. Les mots absents du vocabulaire sont remplacés par un token spécial [UNK] ou <unk>, ce qui fait perdre du sens au texte.

1 Step 1: Tokenization

Voyons quelques exemples d'algorithmes de tokénisation :

2. Tokenizer basé sur les caractères

Les tokenizers basés sur les caractères divisent le texte en caractères, plutôt qu'en mots.

Cela présente deux avantages principaux :

- le vocabulaire est beaucoup plus petit
- il y a beaucoup moins de tokens hors vocabulaire (inconnus) puisque chaque mot peut être construit à partir de caractères.

Let's do tokenization !

L	e	t	'	s	d	o	t	o	k	e	n	i	z	a	t	i	o	n	!

1 Step 1: Tokenization

Voyons quelques exemples d'algorithmes de tokénisation :

2. Tokenizer basé sur les caractères

- Cette approche n'est pas non plus parfaite. Puisque la représentation est maintenant basée sur des caractères plutôt que sur des mots, on pourrait dire intuitivement qu'elle est moins significative : chaque caractère ne signifie pas grand-chose en soi, alors que c'est le cas pour les mots.
- Un autre élément à prendre en compte est que nous nous retrouverons avec une très grande quantité de tokens à traiter par notre modèle. Alors qu'avec un tokenizer basé sur les mots, pour un mot donné on aurait qu'un seul token, avec un tokenizer basé sur les caractères, cela peut facilement se transformer en 10 tokens voire plus.

Pour obtenir le meilleur des deux mondes, nous pouvons utiliser une troisième technique qui combine les deux approches : *la tokénisation en sous-mots.*

1

Step 1: Tokenization

Voyons quelques exemples d'algorithmes de tokénisation :

3. Tokenizer en sous-mots

Les algorithmes de tokenisation en sous-mots reposent sur le principe selon lequel les mots fréquemment utilisés ne doivent pas être divisés en sous-mots plus petits, mais les mots rares doivent être décomposés en sous-mots significatifs.

Let's do tokenization !

Let's </w>	do</w>	token	ization</w>	!</w>

1

Step 1: Tokenization

Voyons quelques exemples d'algorithmes de tokénisation :

3. Tokenizer en sous-mots

Il existe de nombreuses autres techniques. Pour n'en citer que quelques-unes :

- le **Byte-Pair Encoding BPE** utilisé par exemple dans le GPT-2
- le **WordPiece** utilisé par exemple dans BERT
- **SentencePiece** ou **Unigram**, utilisés dans plusieurs modèles multilingues.

Embedding

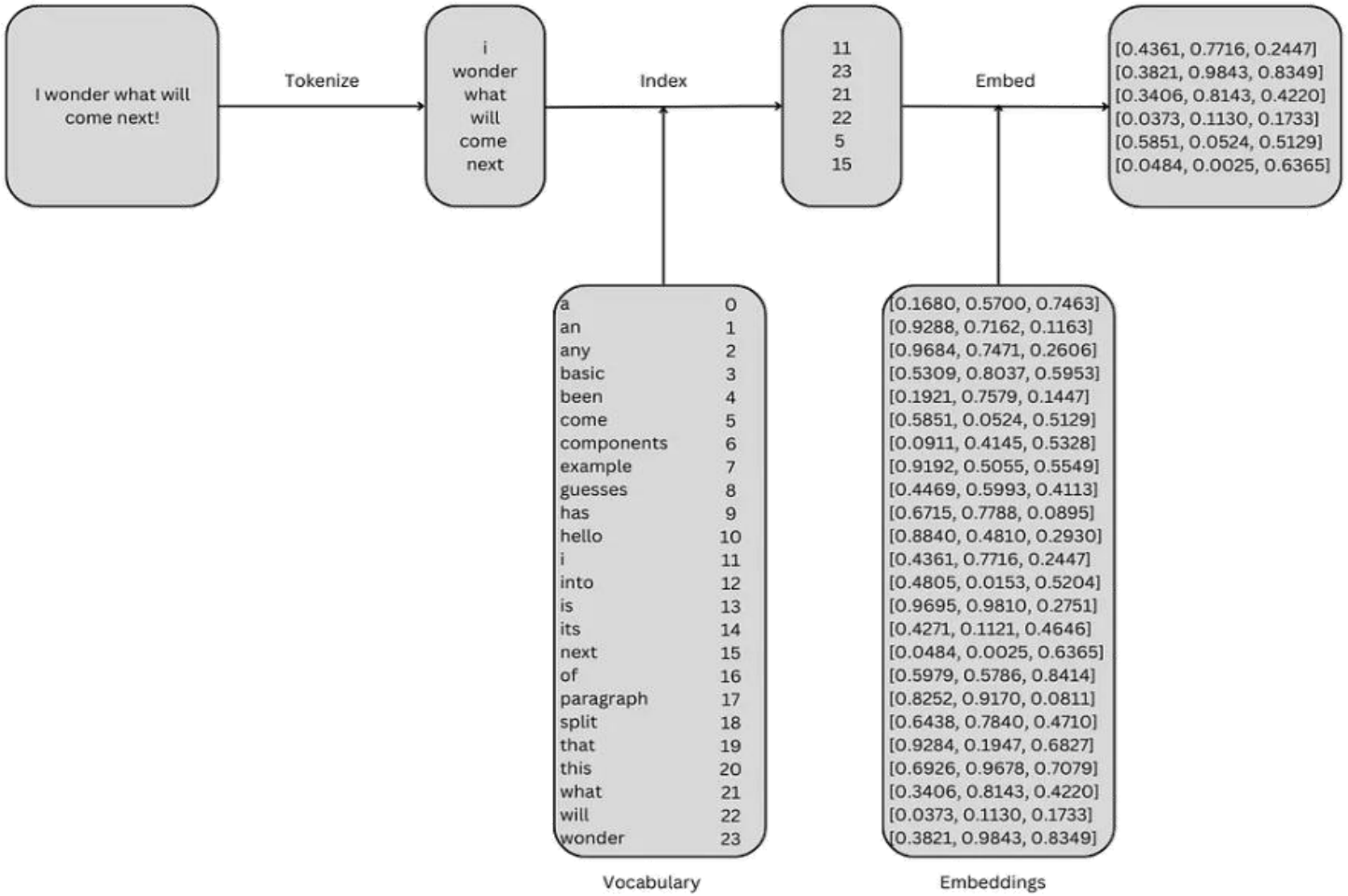
1 Step 1: Tokenization

2 Step 2. Token Embedding

3 Step 3. Positional Encoding

4 Step 4. Final Embedding

Après cette étape, les tokens sont convertis en vecteurs d'embedding à l'aide d'une représentation fixe — un principe similaire à ce que nous avons vu à la séance précédente avec Word2Vec ou d'autres méthodes d'embedding.



Embedding

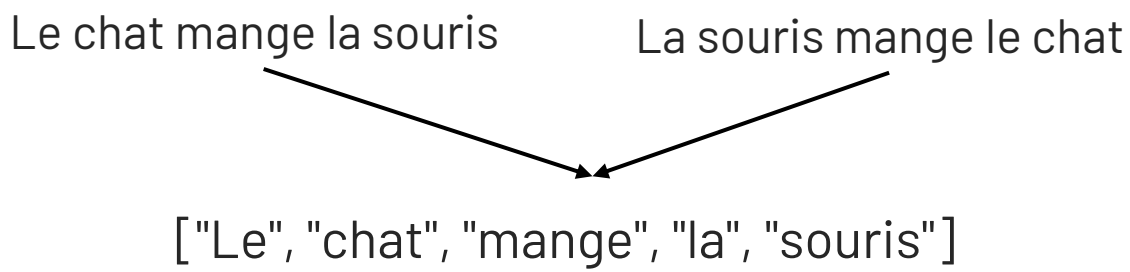
1 Step 1: Tokenization

2 Step 2. Token Embedding

3 Step 3. Positional Encoding

4 Step 4. Final Embedding

Les Transformers reçoivent tous les tokens en parallèle (grâce à l'attention), mais cela signifie qu'ils ne savent pas naturellement dans quel ordre les mots apparaissent.



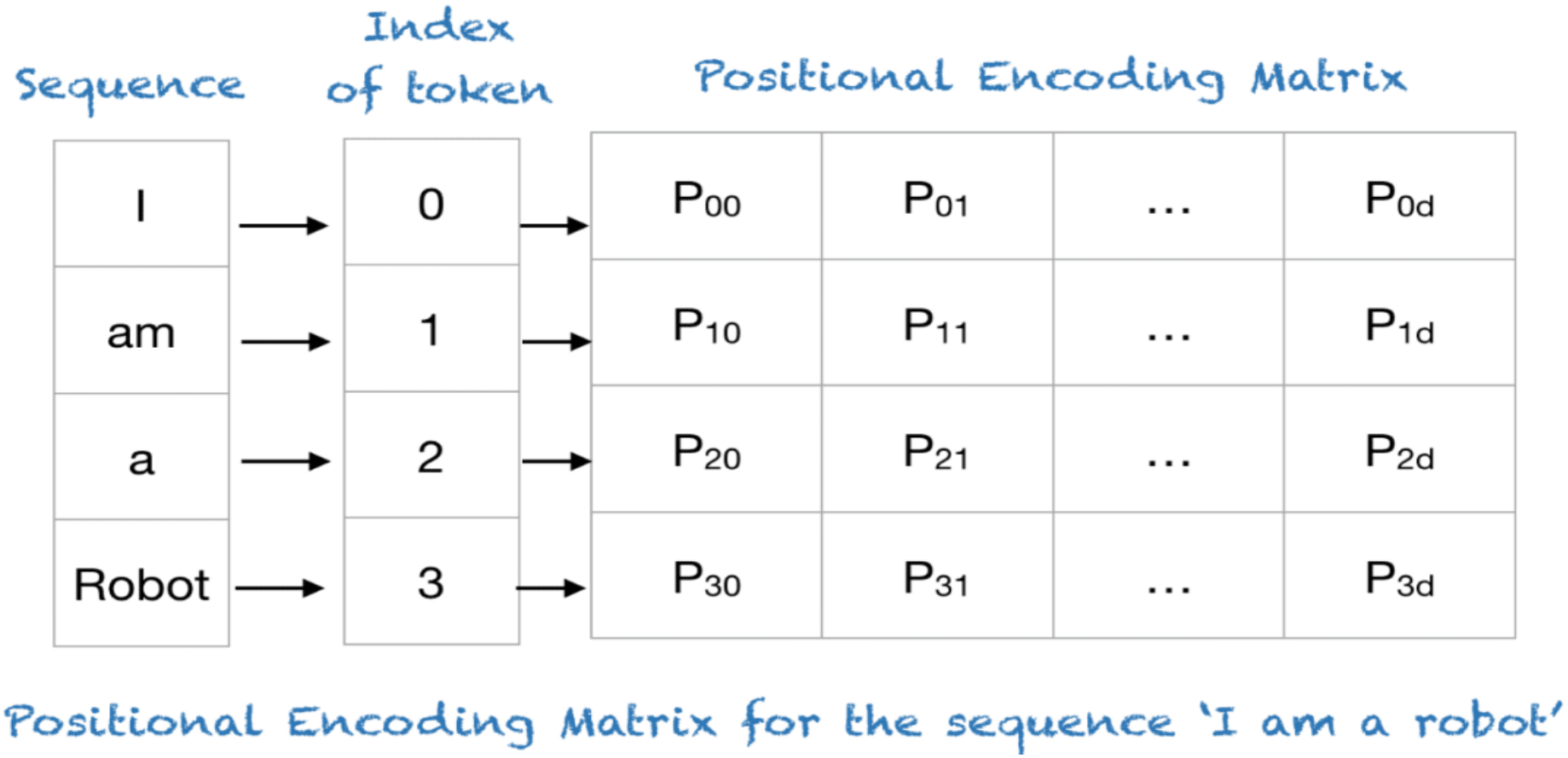
Les deux phrases contiennent les mêmes mots, mais l'ordre change complètement le sens.

Pour donner à chaque mot une **information sur sa position** dans la phrase, on ajoute un **vecteur de position** à son embedding :

$$x_i = E[t_i] + P[i]$$

Embedding

3 Step 3. Positional Encoding



d : la dimension du vecteur d'embedding (embedding dimension)

3

Step 3. Positional Encoding

Supposons que vous ayez une séquence d'entrée de longueur LL et que vous souhaitiez représenter la position du pos ème élément dans cette séquence. L'encodage positionnel est défini à l'aide de fonctions sinus et cosinus de fréquences variées :

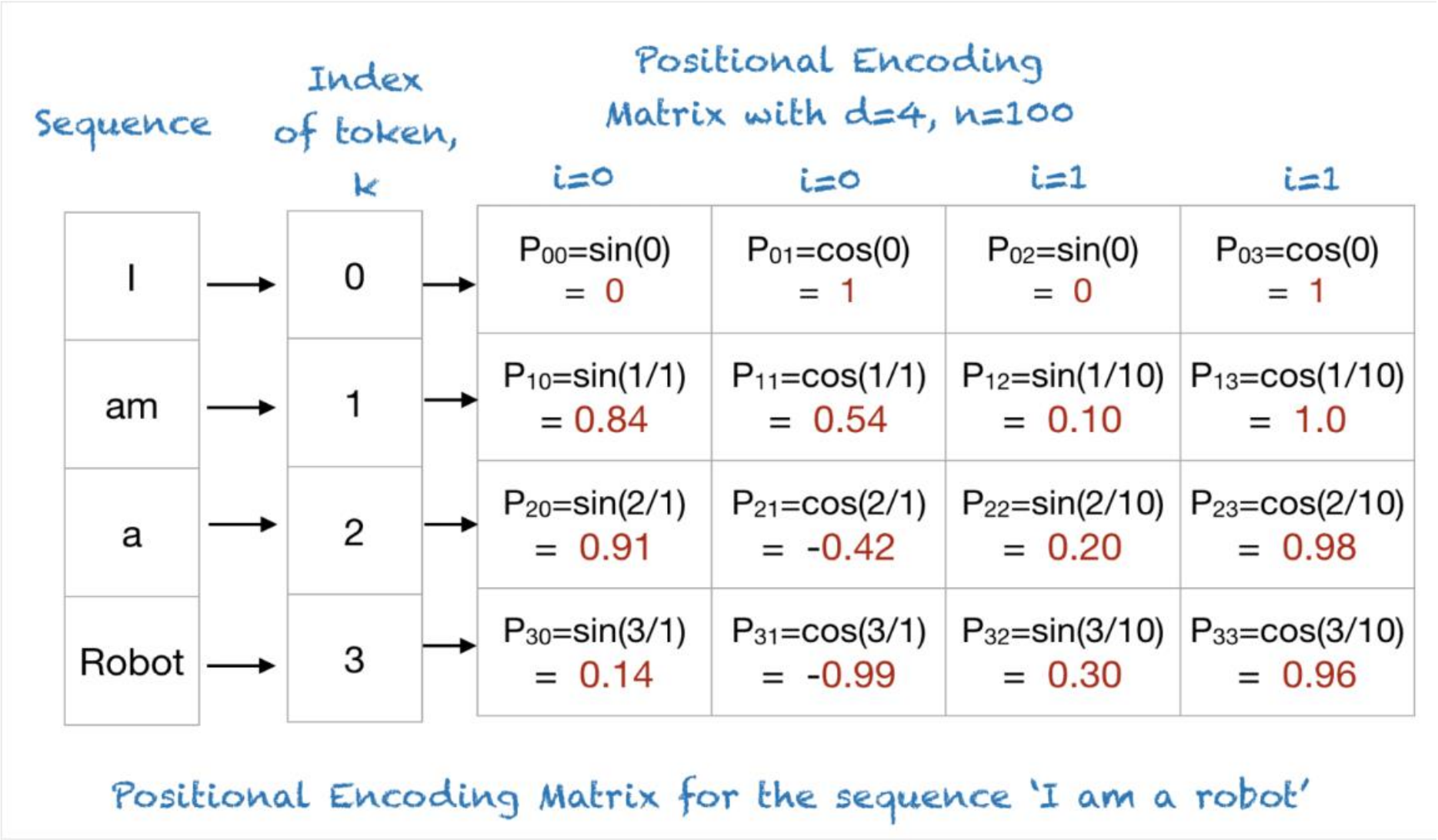
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

- pos : position de l'élément dans la séquence d'entrée, 0≤pos<L/2
- d_model : dimension de l'espace d'embedding de sortie
- i : indice des colonnes, 0≤i<d/2; chaque valeur de i est utilisée à la fois pour les fonctions sinus et cosinus

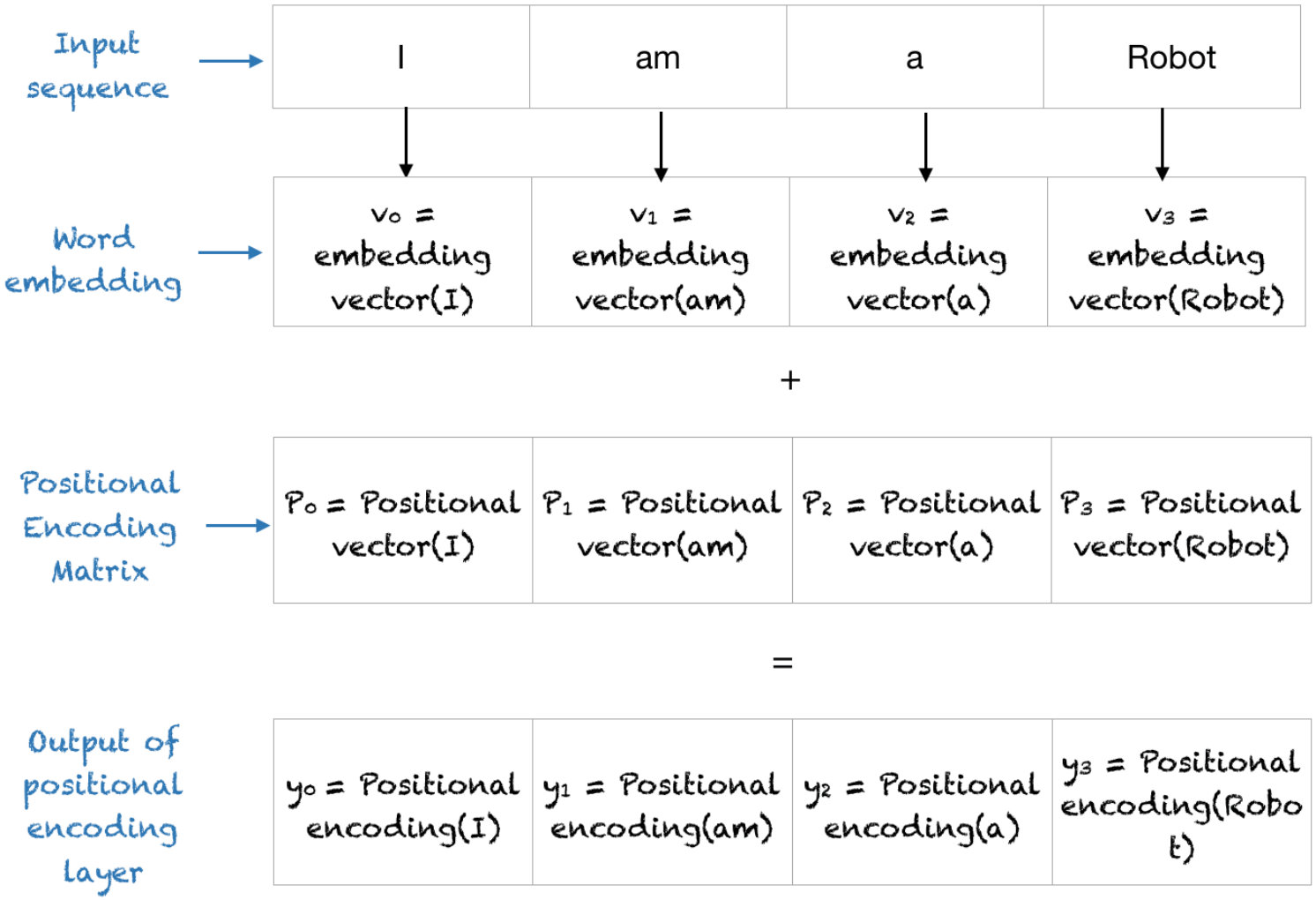
Embedding

3 Step 3. Positional Encoding

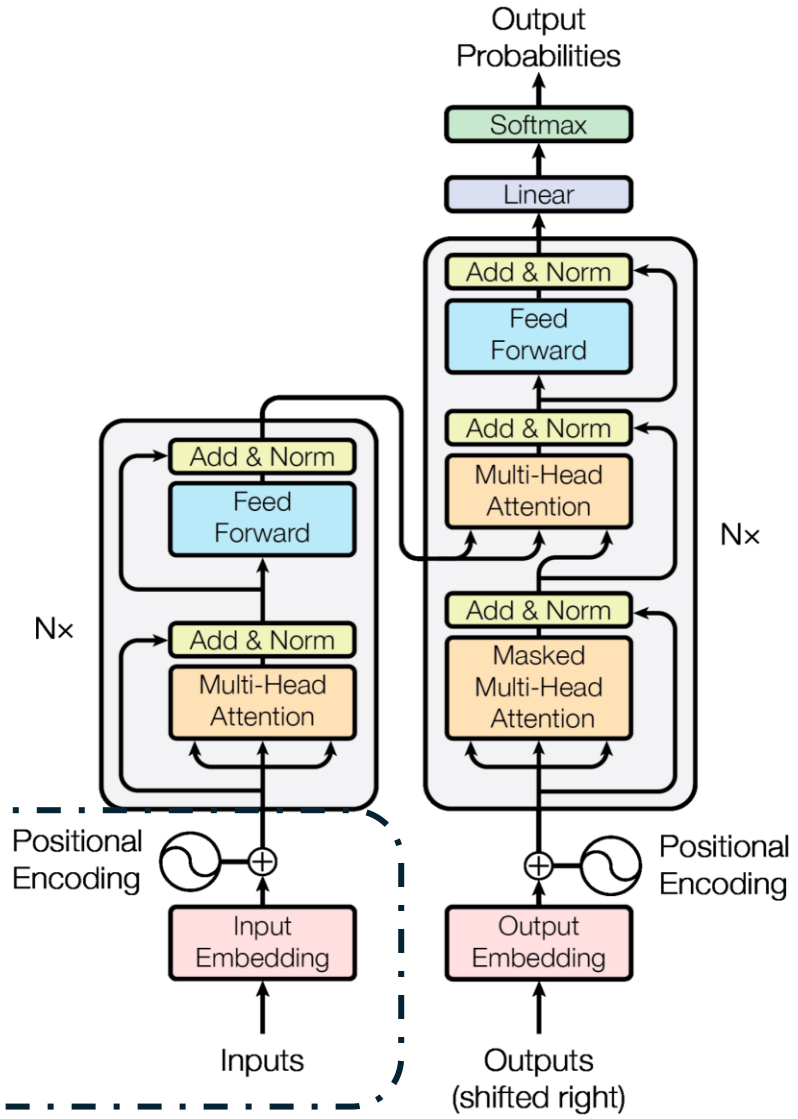


Embedding

1. Step 1: Tokenization
2. Step 2. Token Embedding
3. Step 3. Positional Encoding
4. Step 4. Final Embedding



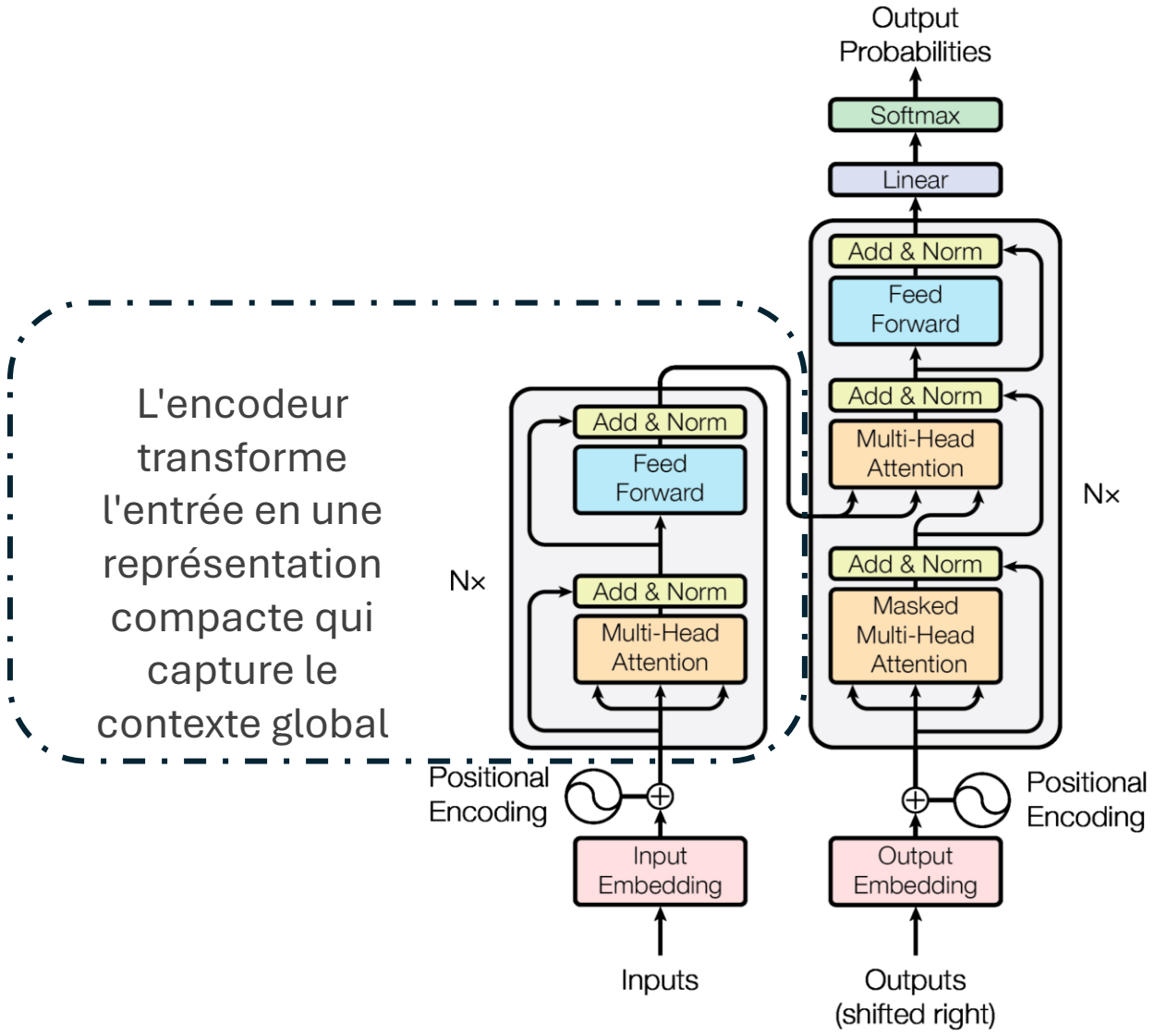
Transformers



L'Embedding Layer avec Positional Encoding transforme la séquence d'entrée en une représentation vectorielle continue intégrant à la fois le sens des tokens et leur position dans la phrase.

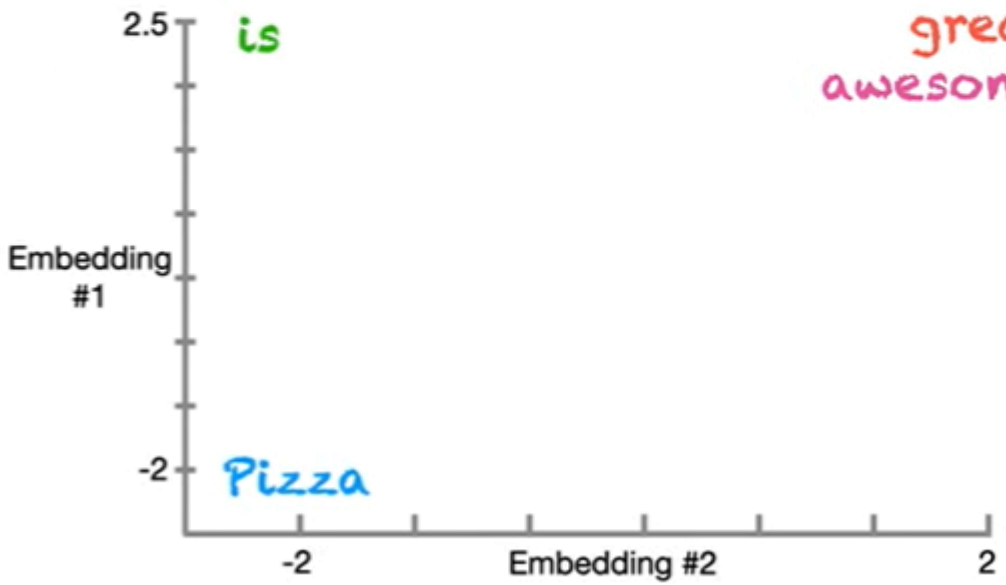
Transformers

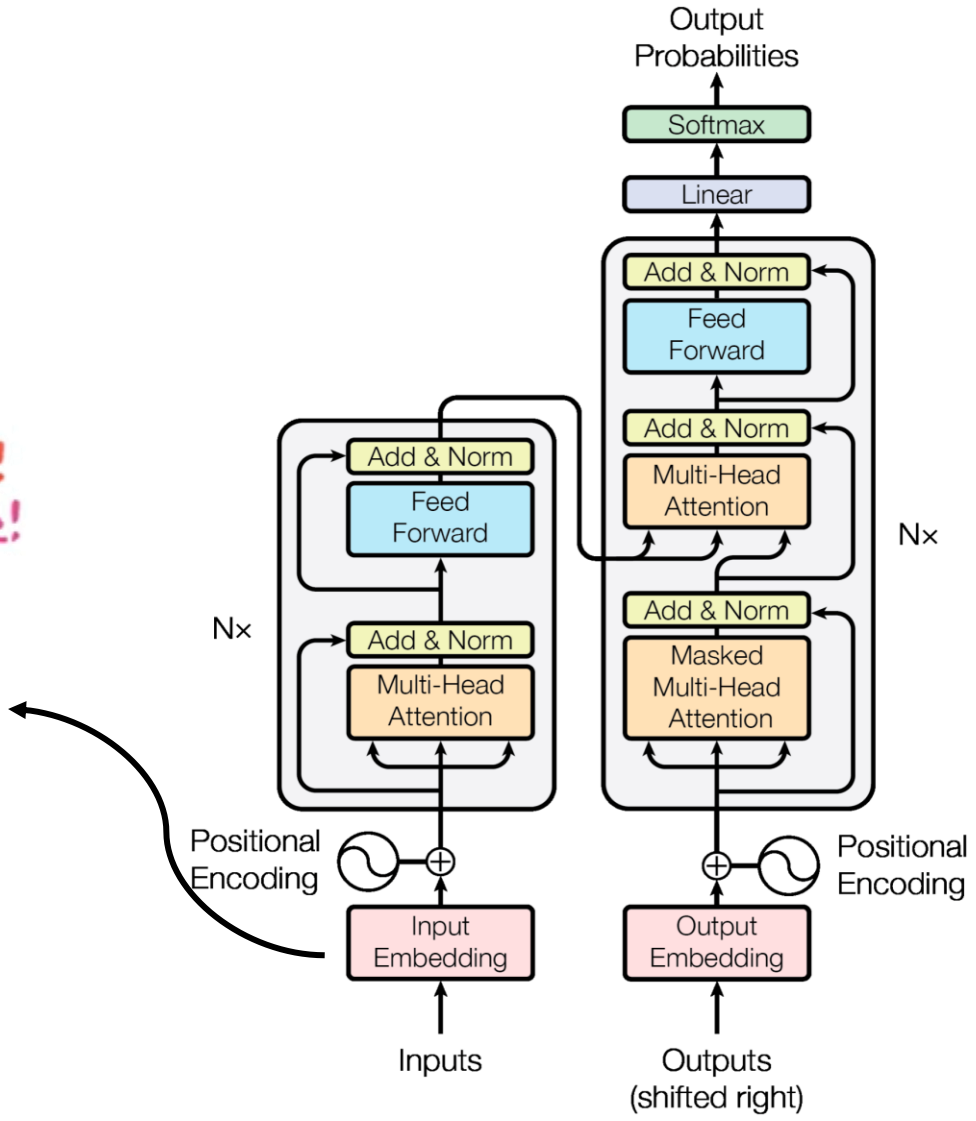
L'encodeur transforme l'entrée en une représentation compacte qui capture le contexte global



Transformers

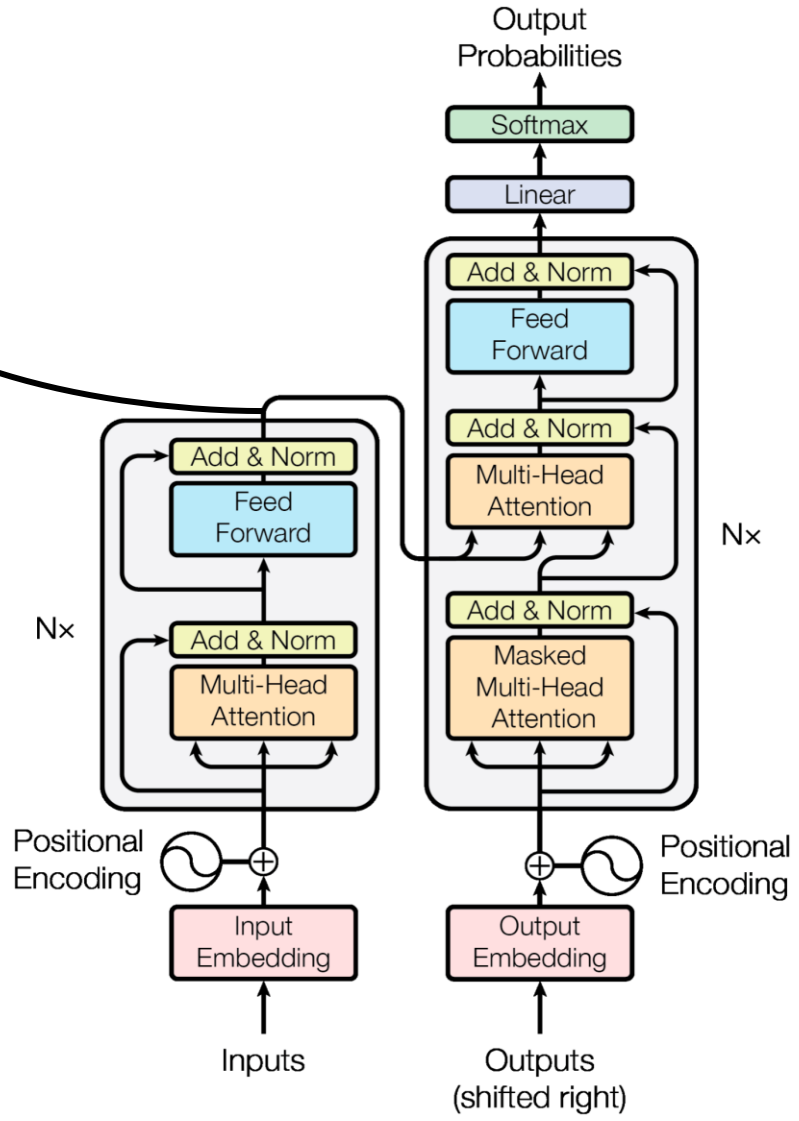
And just like plain old **Word Embeddings** can help cluster similar words that are used in similar ways…



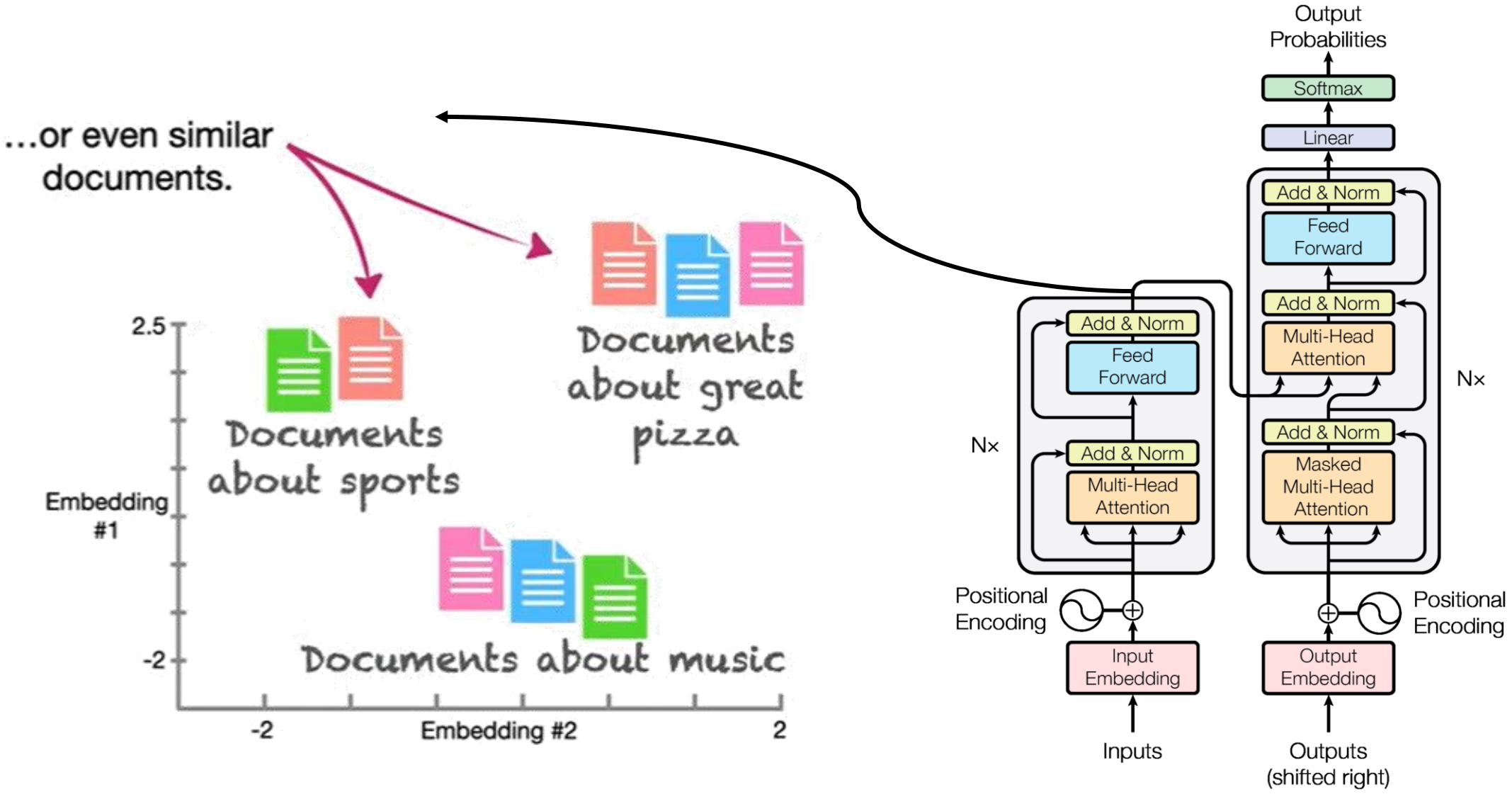


Transformers

Context Aware Embeddings can help cluster similar sentences…



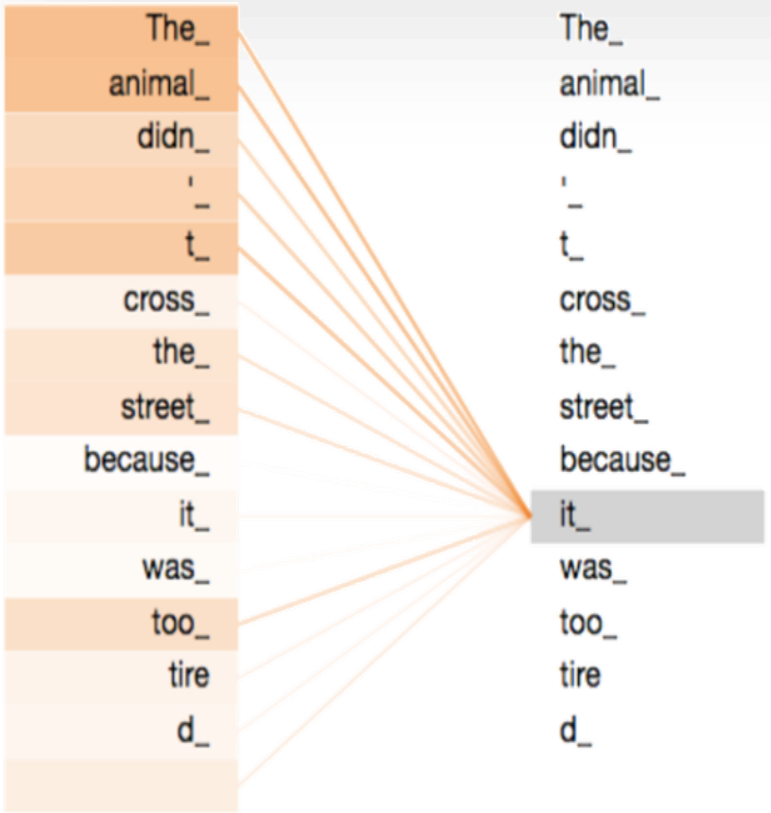
Transformers



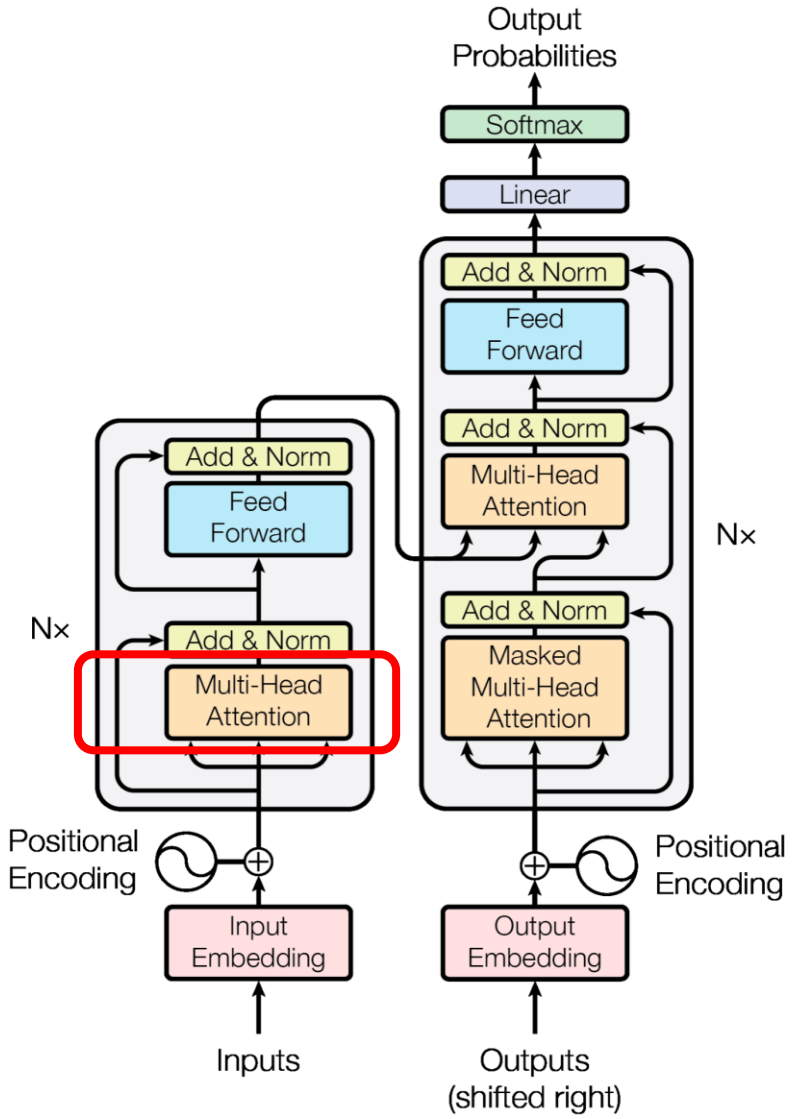
Transformers

Transformers applique un mécanisme d'attention.

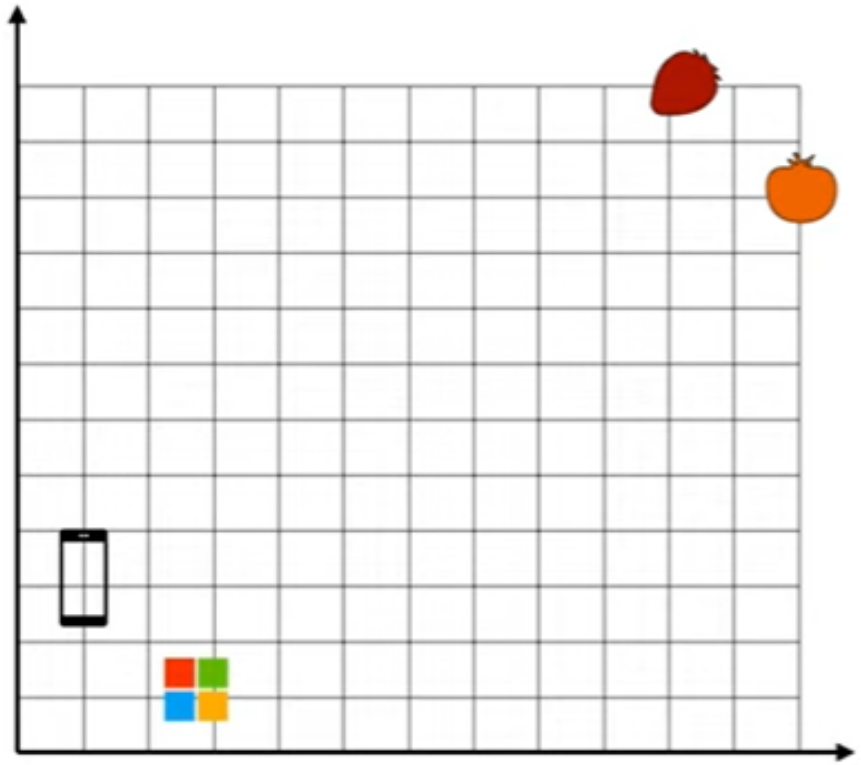
"The animal didn't cross the street because it was too tired."



Ce mécanisme d'attention permet à chaque token de la séquence de prêter attention à tous les autres tokens, ce qui capture les dépendances à long terme.



Embedding Quiz



Top right or bottom left?

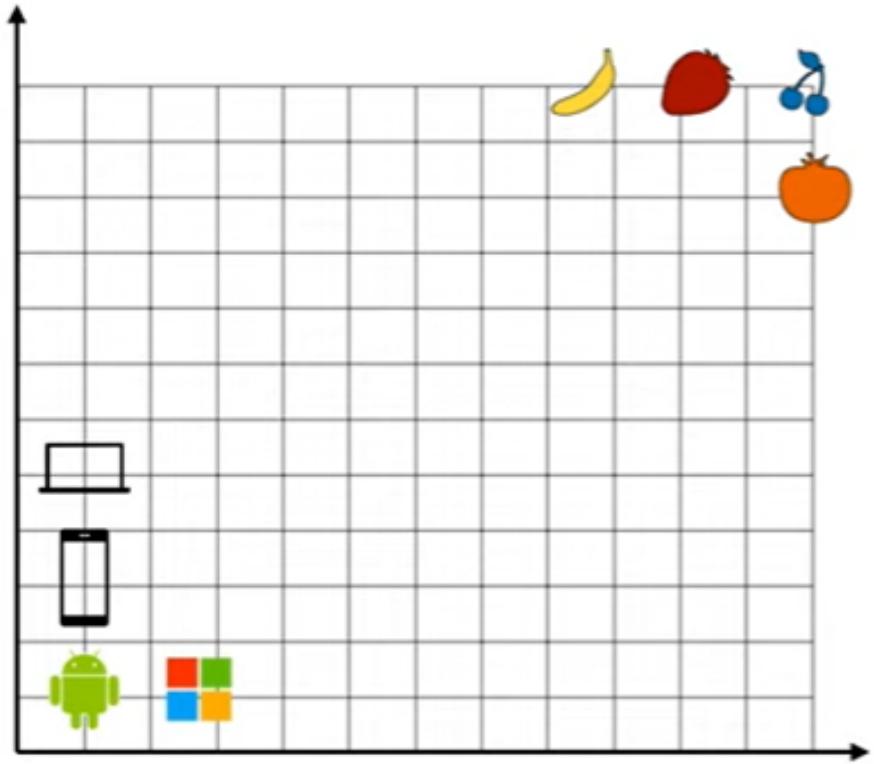
Cherry

Android

Laptop

Banana

Embedding Quiz



Top right or bottom left?

Cherry

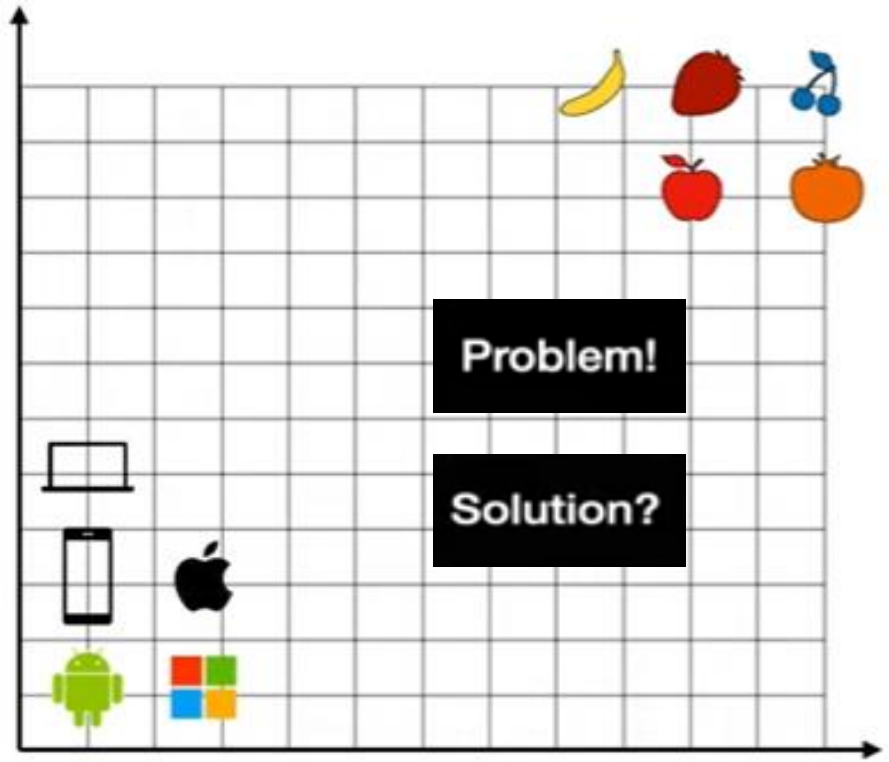
Android

Laptop

Banana

Apple?

Embedding Quiz

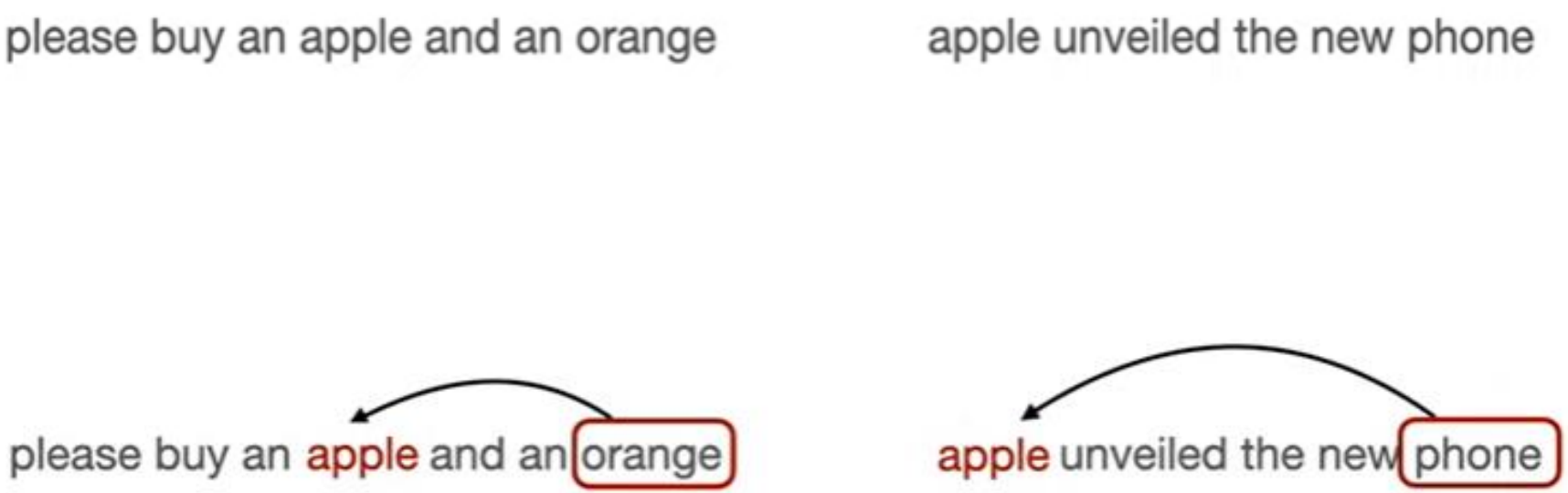


Top right or bottom left?

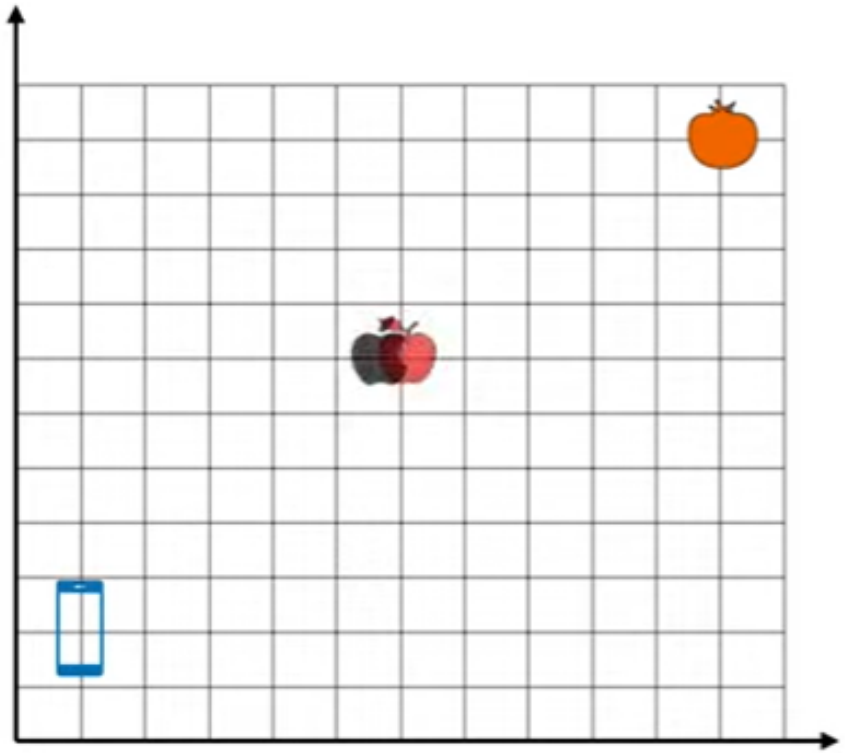
Cherry
Android
Laptop
Banana
Apple?

Solution : Attention

Attention


please buy an apple and an orange
apple unveiled the new phone
please buy an apple and an orange
apple unveiled the new phone

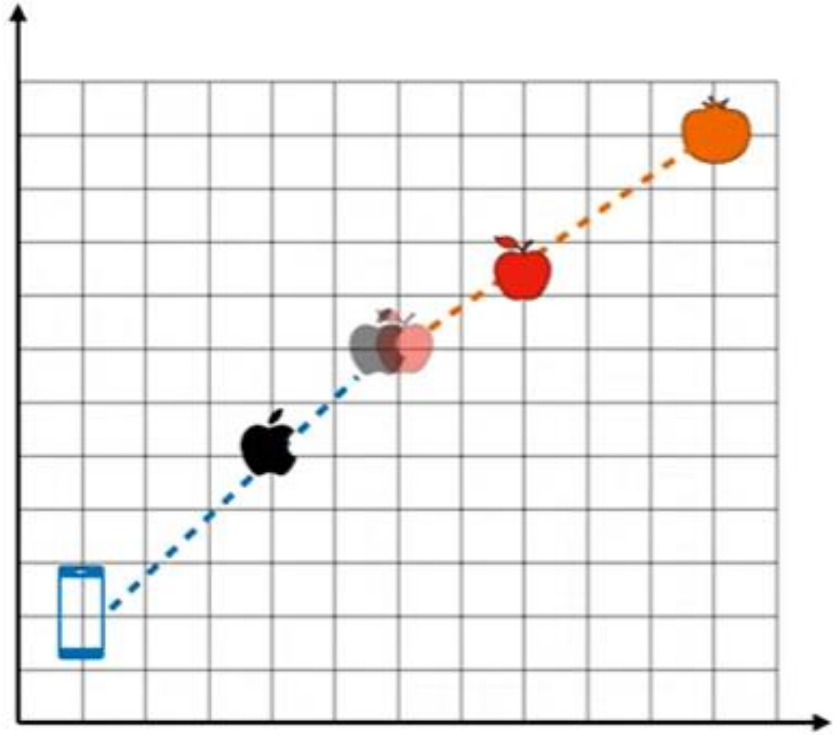
Attention



please buy an apple and an orange

apple unveiled the new phone

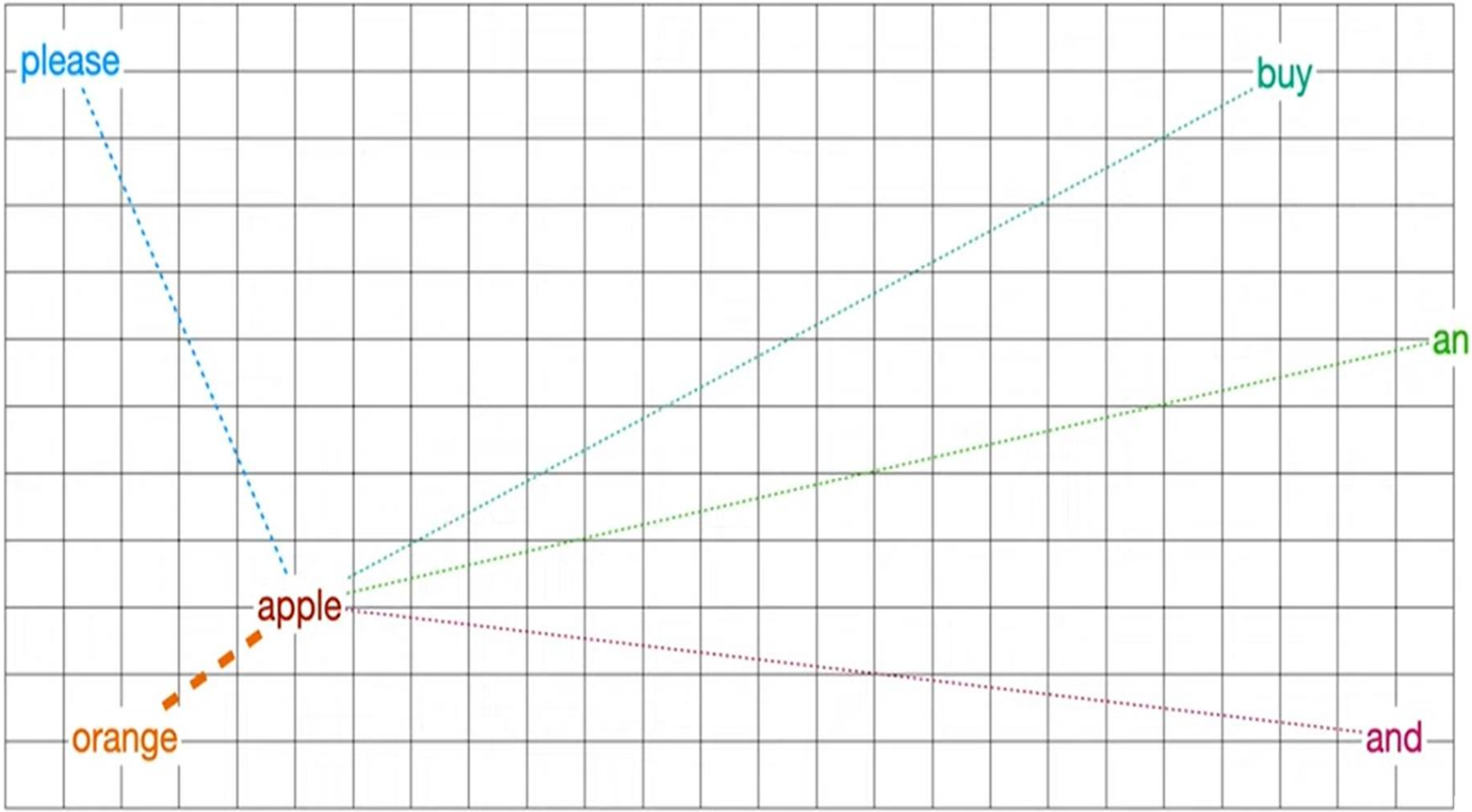
Attention




please buy an apple and an orange
apple unveiled the new phone

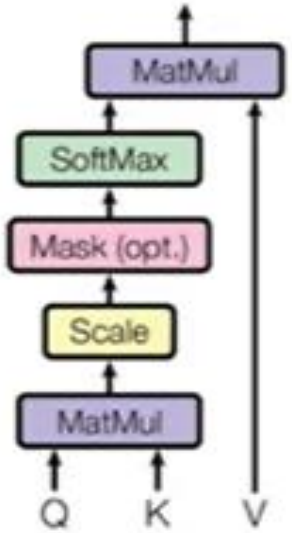
Attention

Et les autres mots ?

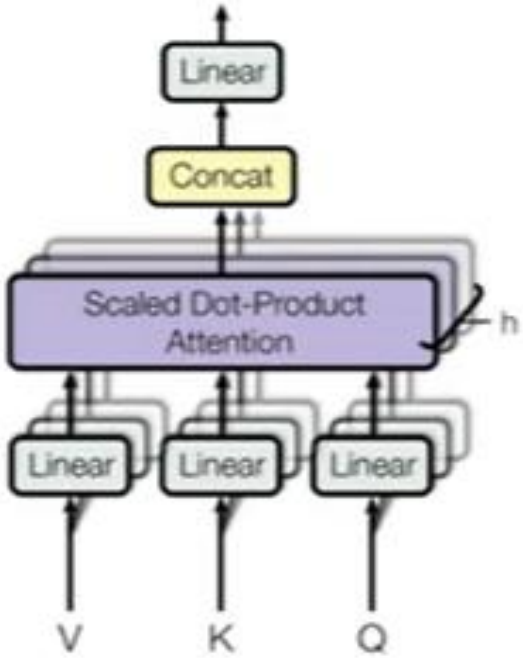


Attention

Scaled Dot-Product Attention



Multi-Head Attention



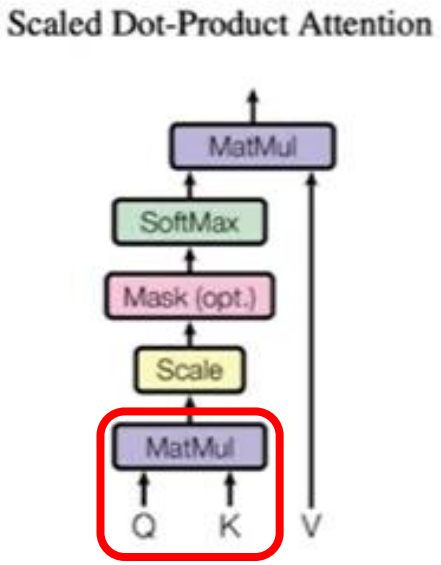
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

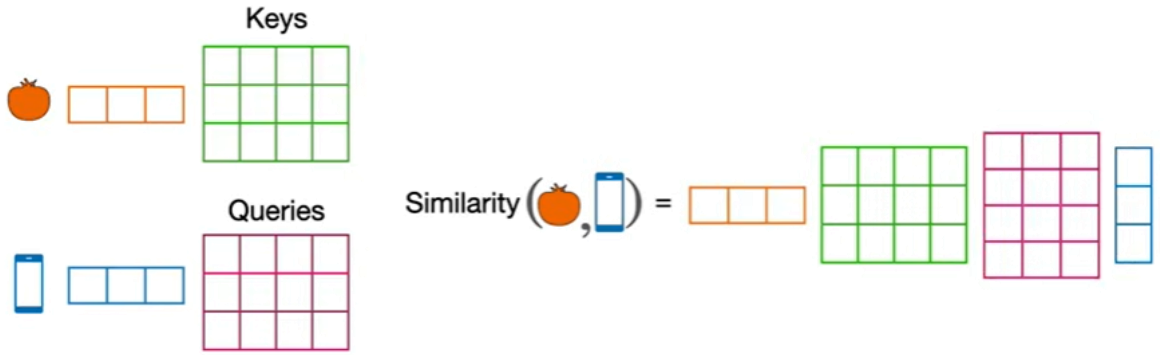
$$\mathbf{W_Q} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad \mathbf{W_K} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad \mathbf{W_V} \in \mathbb{R}^{d_{\text{model}} \times d_k}$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Self Attention



le calcul d'attention dans les Transformers repose **sur une mesure de similarité** entre les **requêtes (queries)** et les **clés (keys)**.



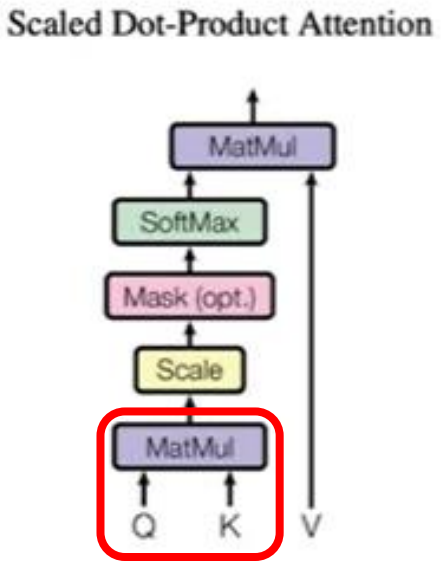
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

$$\mathbf{W_Q} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad \mathbf{W_K} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad \mathbf{W_V} \in \mathbb{R}^{d_{\text{model}} \times d_k}$$

Attention

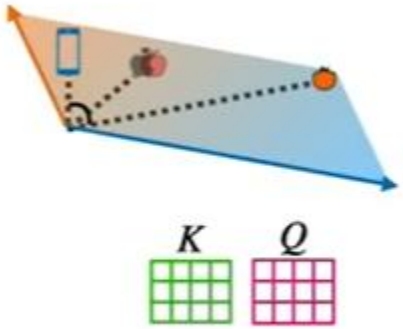
Self Attention



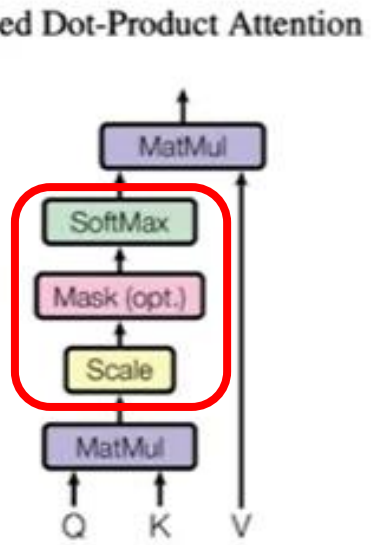
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

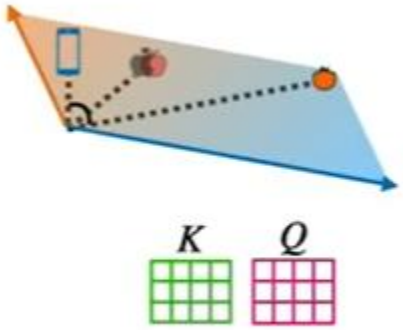
$$\mathbf{W_Q} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad \mathbf{W_K} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad \mathbf{W_V} \in \mathbb{R}^{d_{\text{model}} \times d_k}$$



Self Attention



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

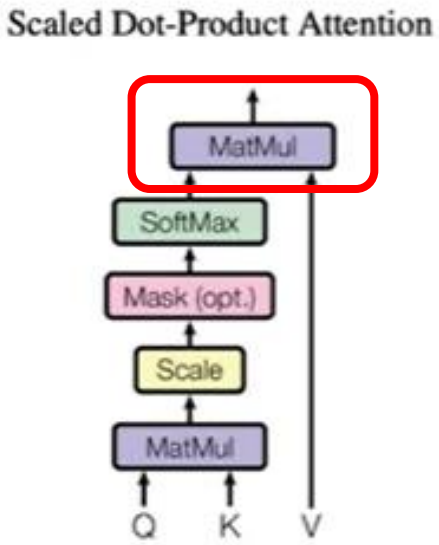


$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

$$\mathbf{W_Q} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad \mathbf{W_K} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad \mathbf{W_V} \in \mathbb{R}^{d_{\text{model}} \times d_k}$$

Attention

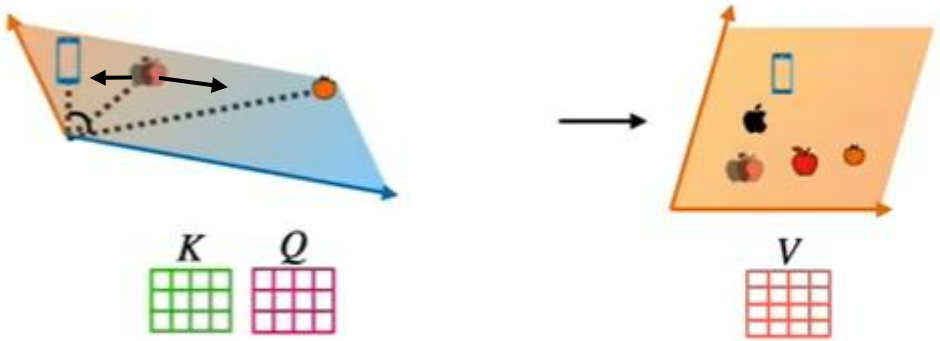
Self Attention



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

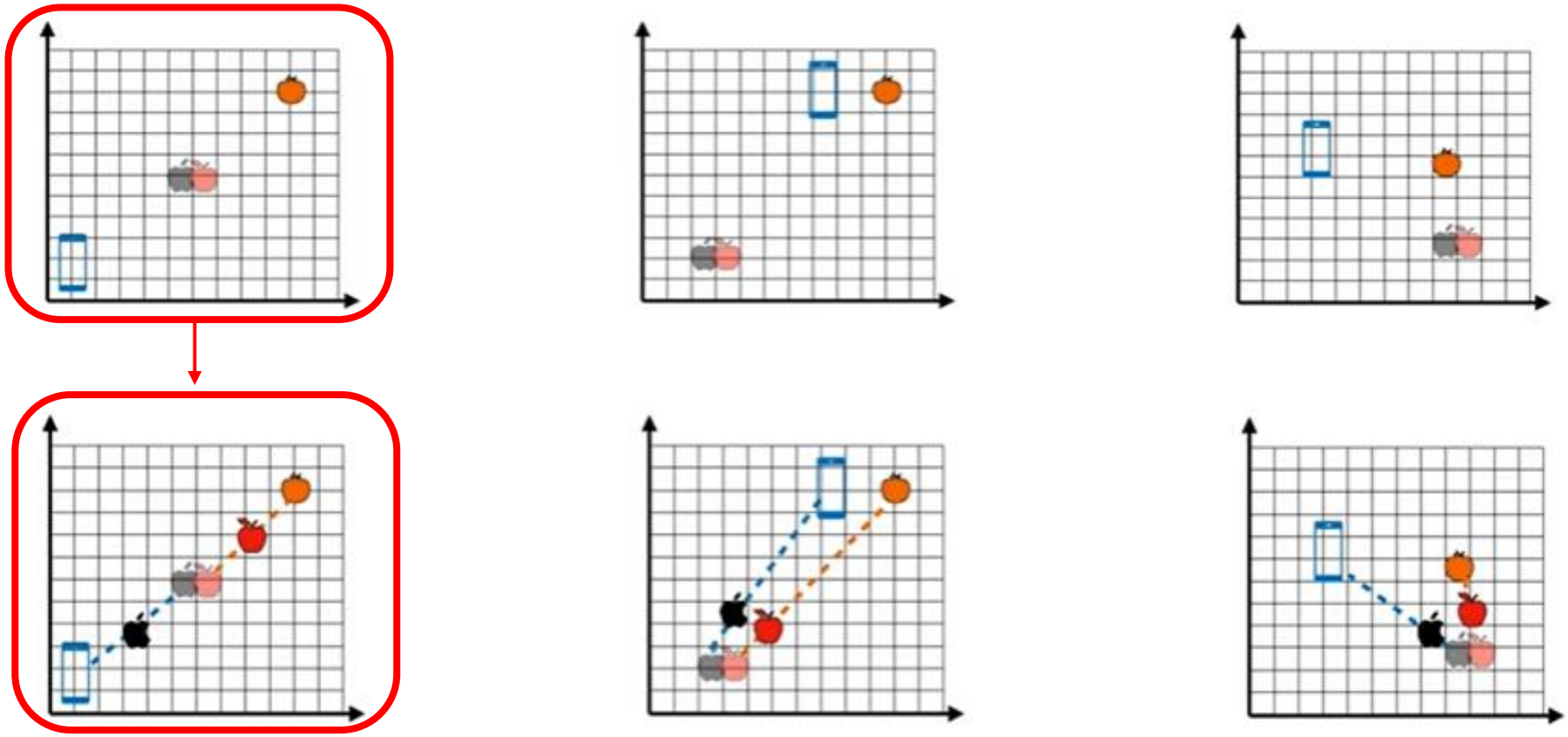
$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

$$\mathbf{W_Q} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad \mathbf{W_K} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad \mathbf{W_V} \in \mathbb{R}^{d_{\text{model}} \times d_k}$$



Une seule représentation est-elle suffisante ?

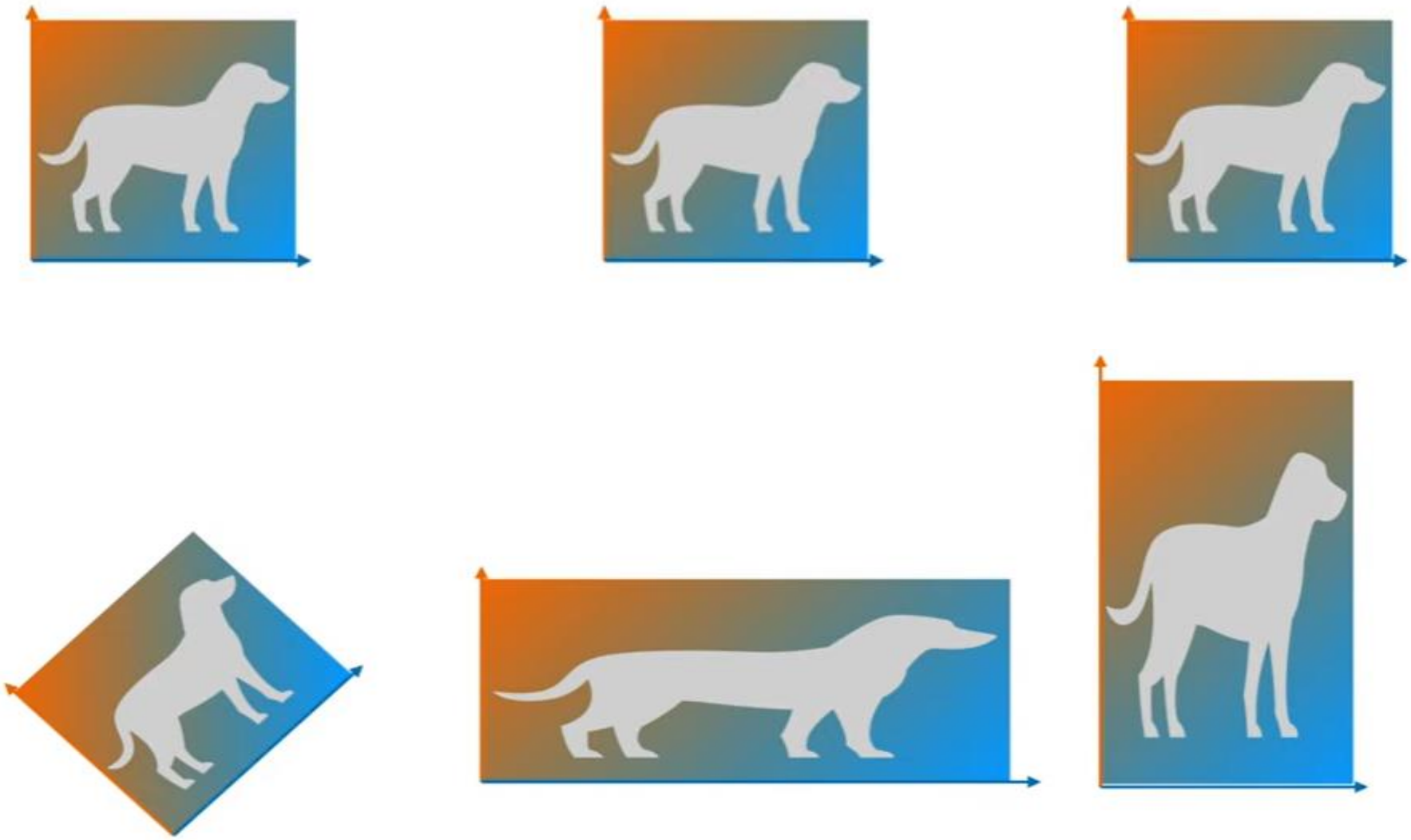
☑ No, Idéalement, nous aimerions avoir beaucoup d'intégration.



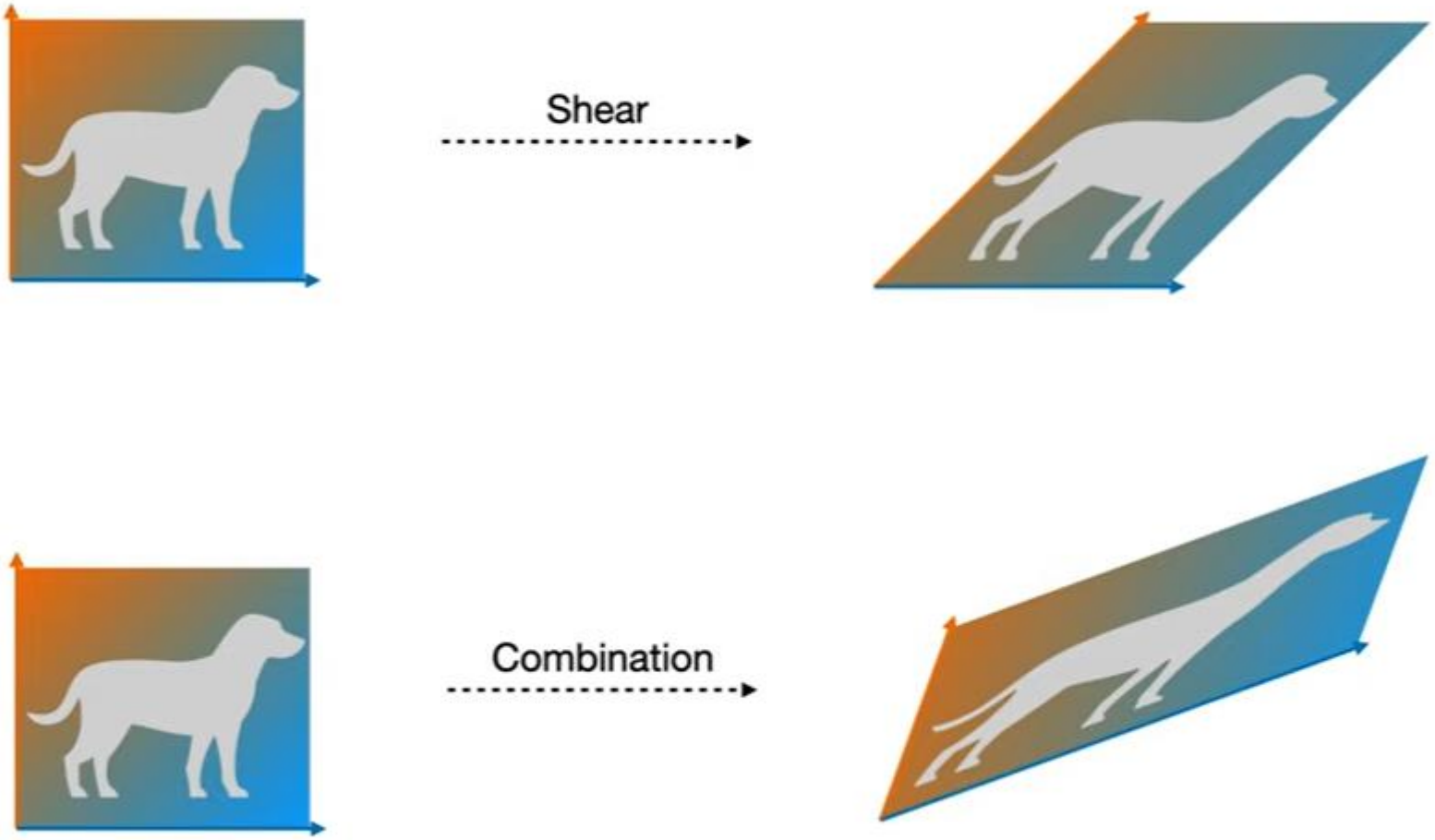
Problème : Construire de nombreux embeddings en modifiant des embeddings existants.

Solution : Nous allons créer de nouveaux embeddings **en modifiant ceux qui existent déjà.**

Les transformations linéaire

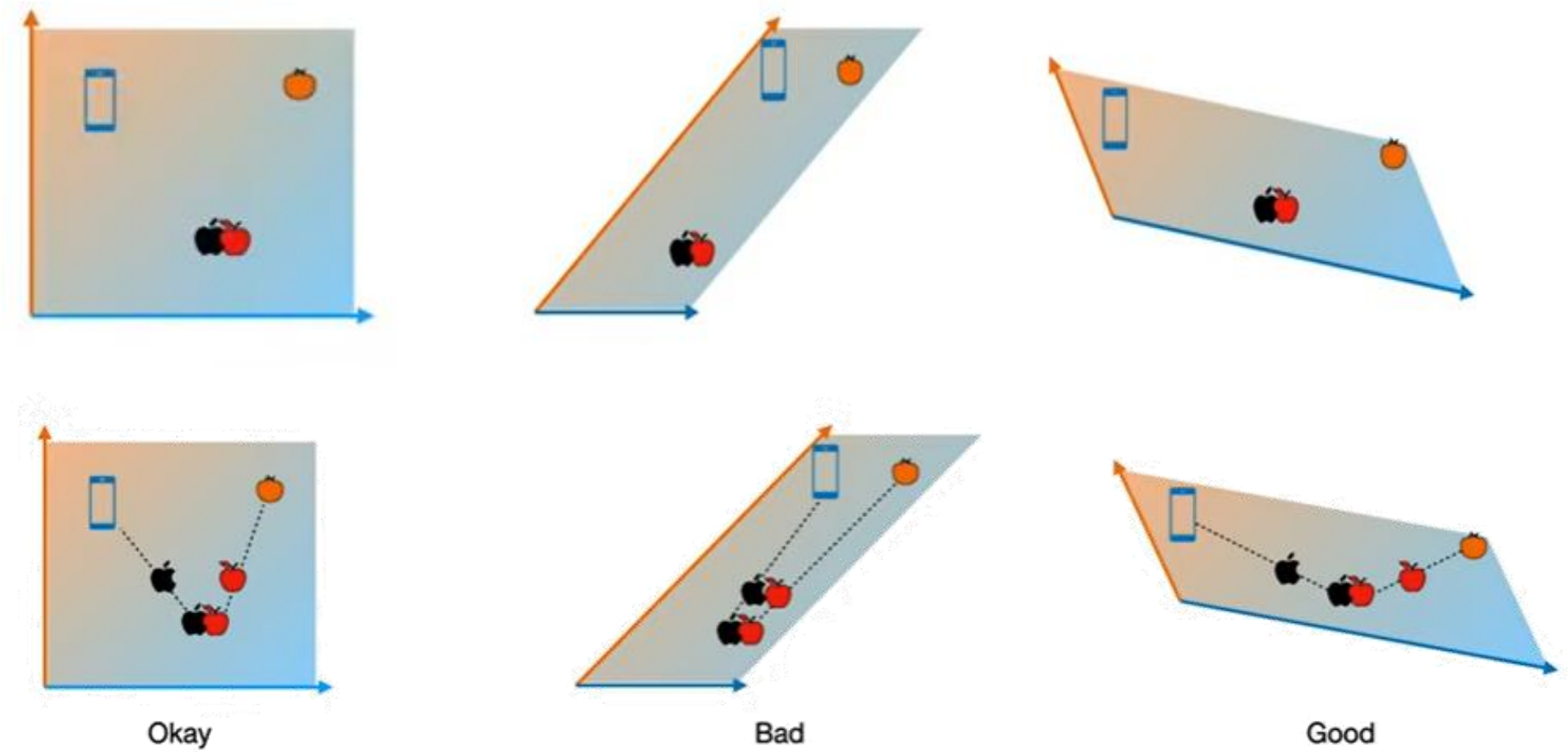


Les transformations linéaire

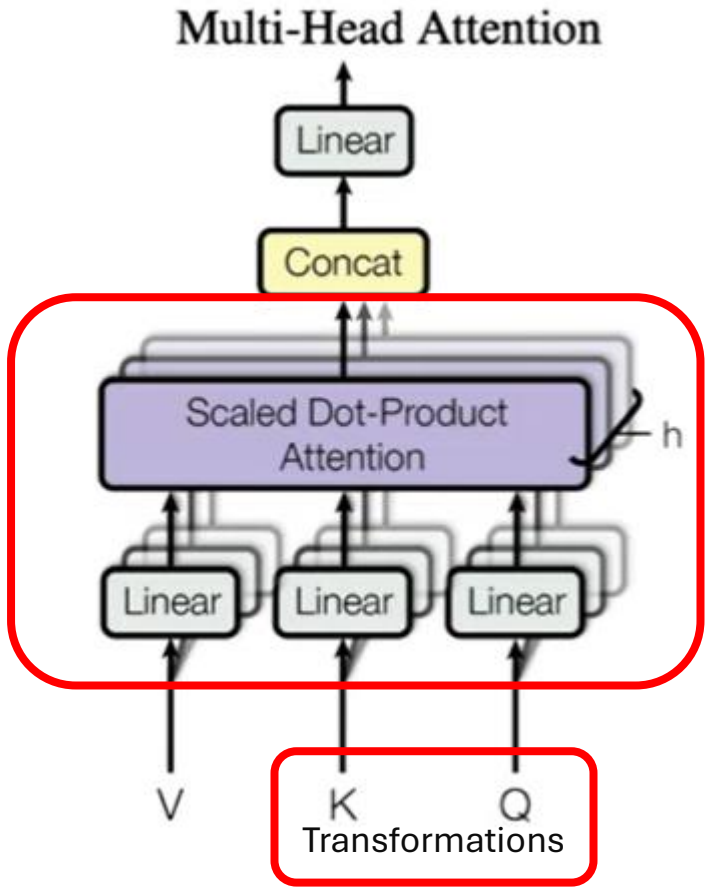


Attention

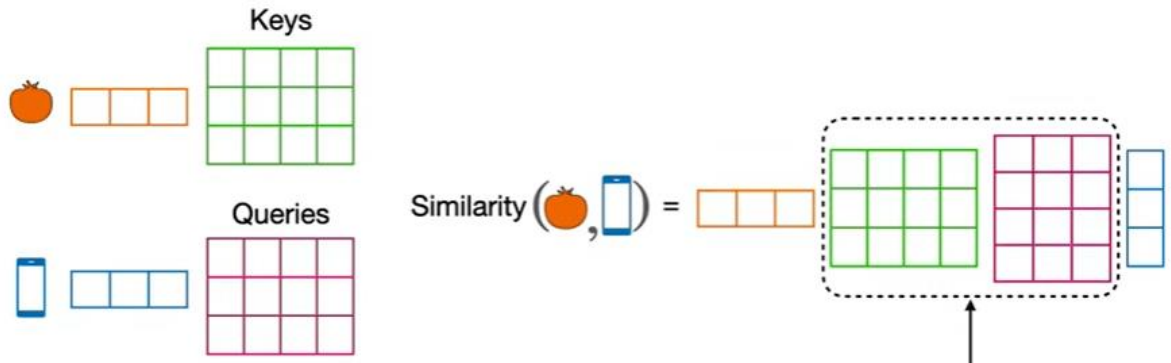
Créer de nouveaux embeddings en modifiant ceux qui existent déjà.

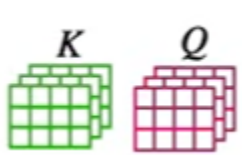


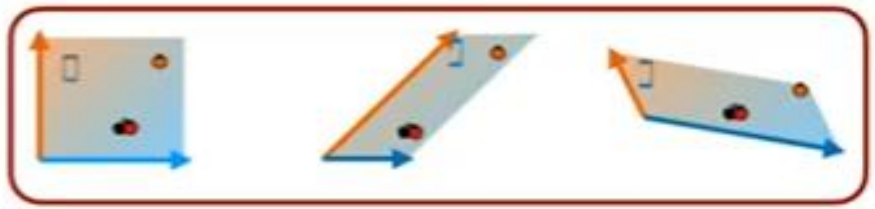
Attention

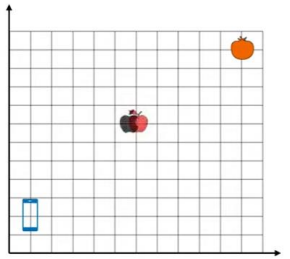


$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$





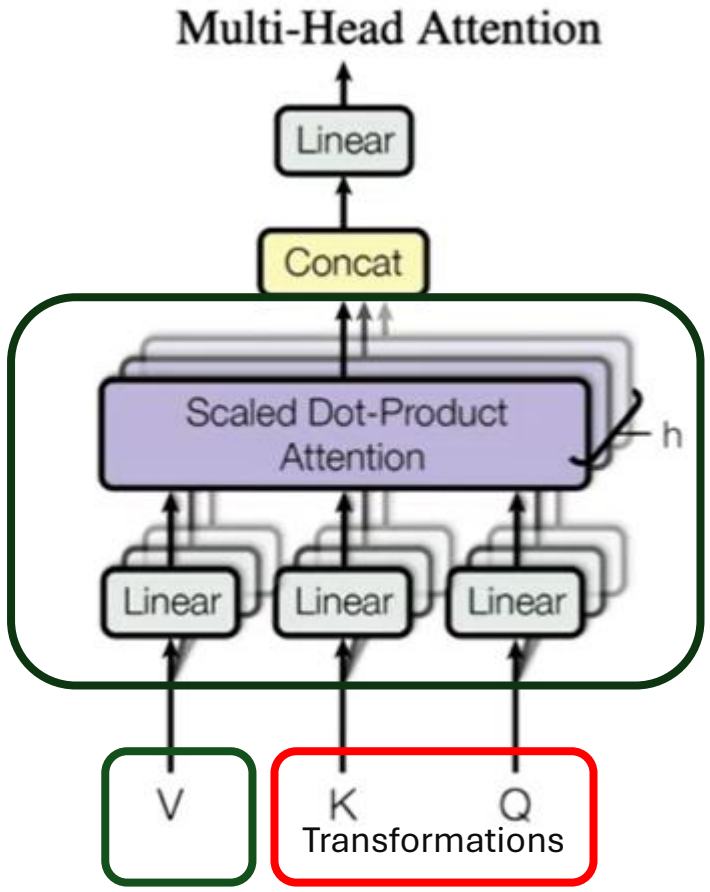




please buy an apple and an orange

apple unveiled the new phone

Attention



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

an apple and an orange

	Orange	Apple	And	An
Orange	0.4	0.3	0.15	0.15
Apple	0.3	0.4	0.15	0.15
And	0.15	0.15	0.5	0.5
An	0.15	0.15	0.5	0.5

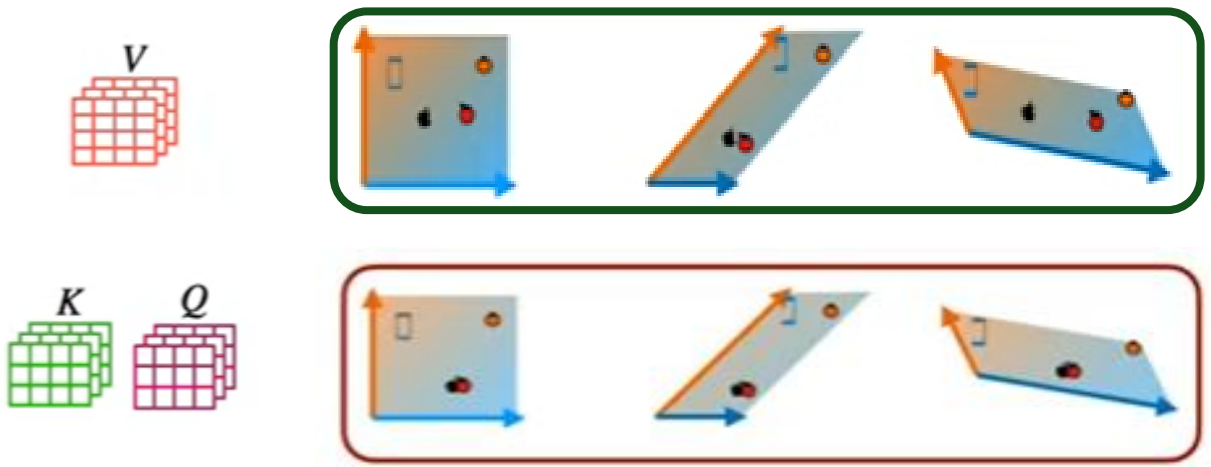
Value matrix

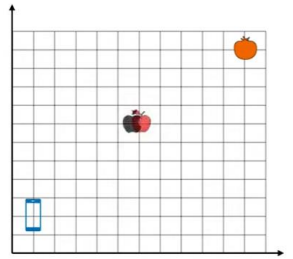
=

	Orange	Apple	And	An
Orange	v_{11}	v_{12}	v_{13}	v_{14}
Apple	v_{21}	v_{22}	v_{23}	v_{24}
And	v_{31}	v_{32}	v_{33}	v_{34}
An	v_{41}	v_{42}	v_{43}	v_{44}

apple ⟶ 0.3 · orange
+0.4 · apple
+0.15 · and
+0.15 · an

apple ⟶ v_{21} · orange
+v_{22} · apple
+v_{23} · and
+v_{24} · an

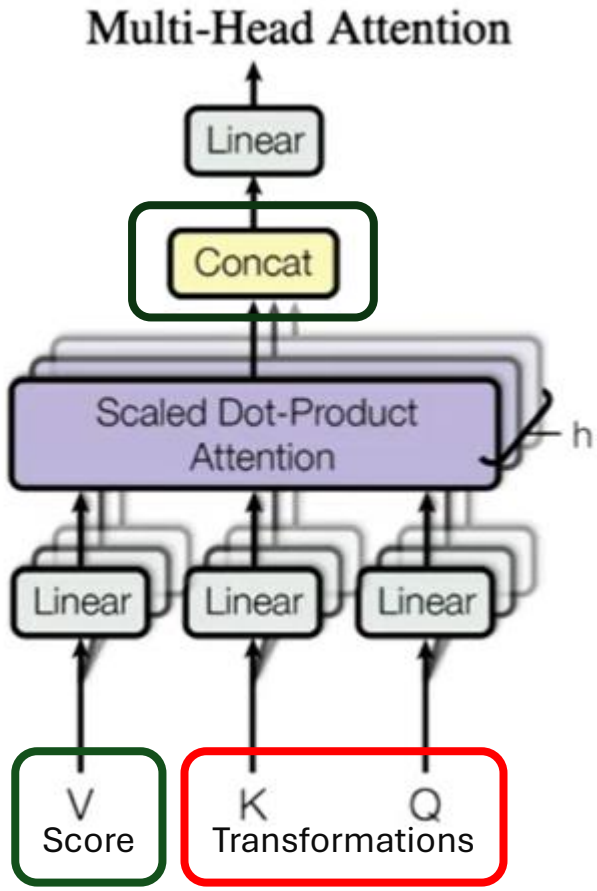




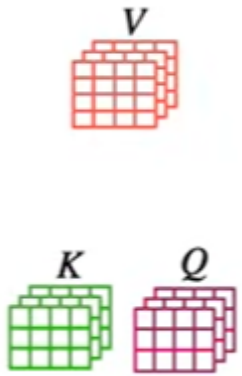
please buy an apple and an orange

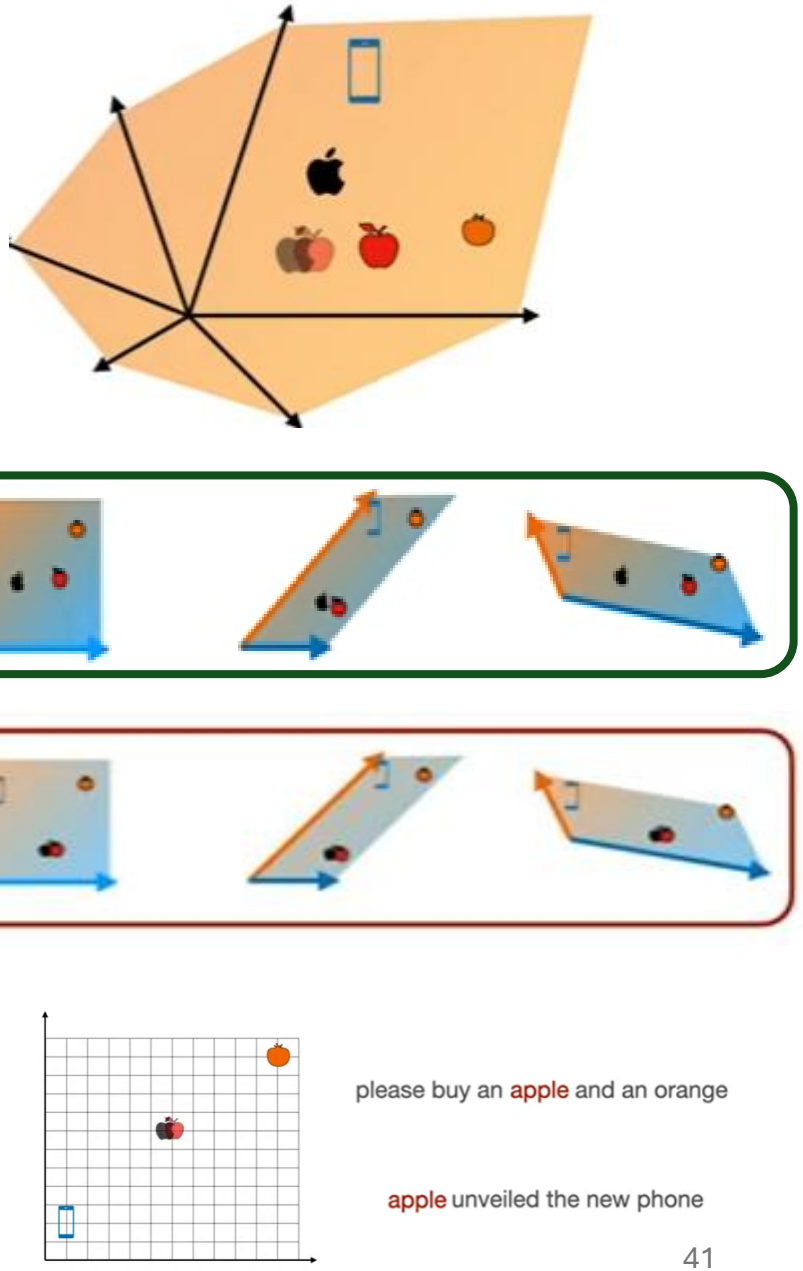
apple unveiled the new phone

Attention

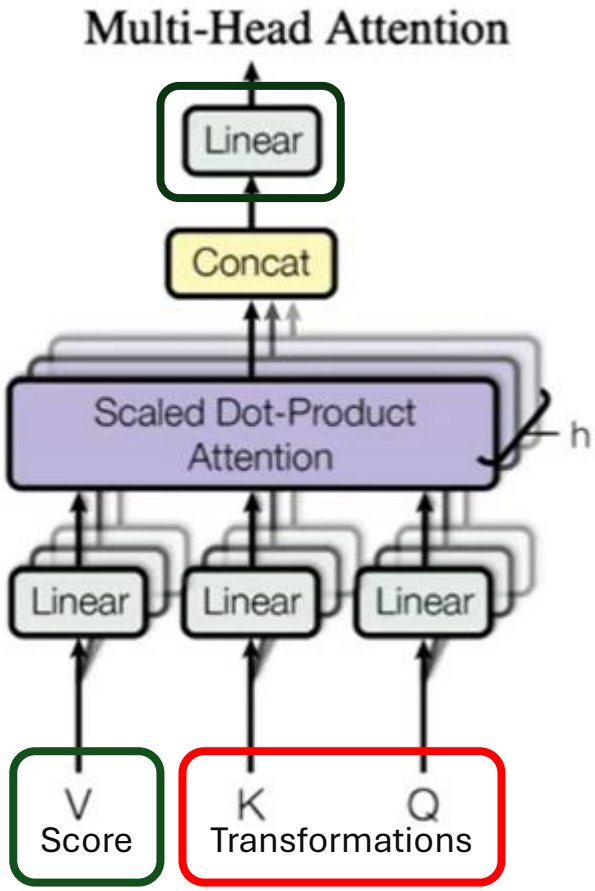


$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

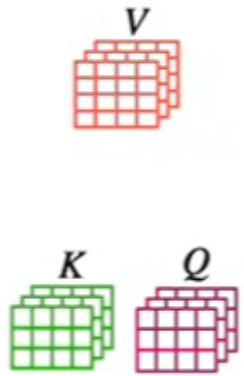


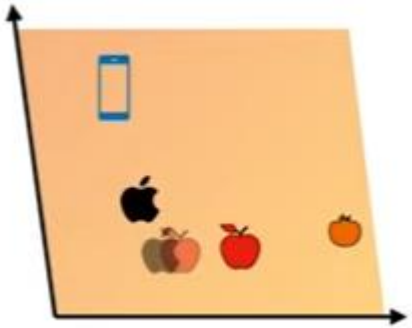


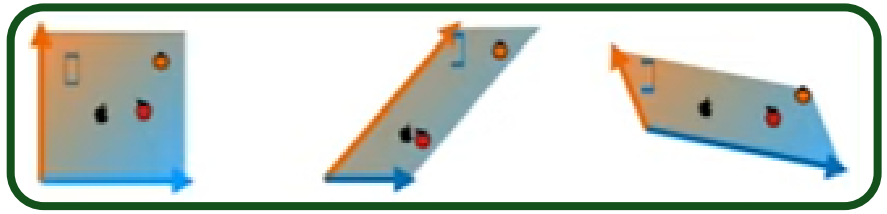
Attention

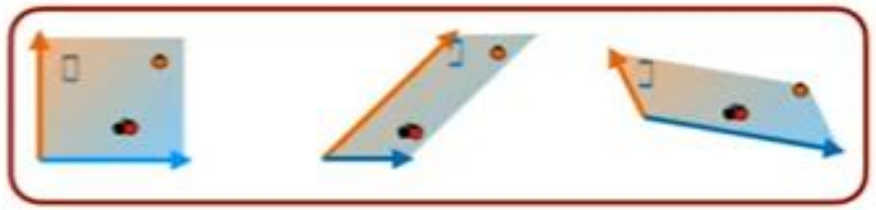


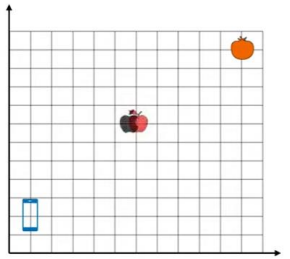
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$











please buy an apple and an orange

apple unveiled the new phone

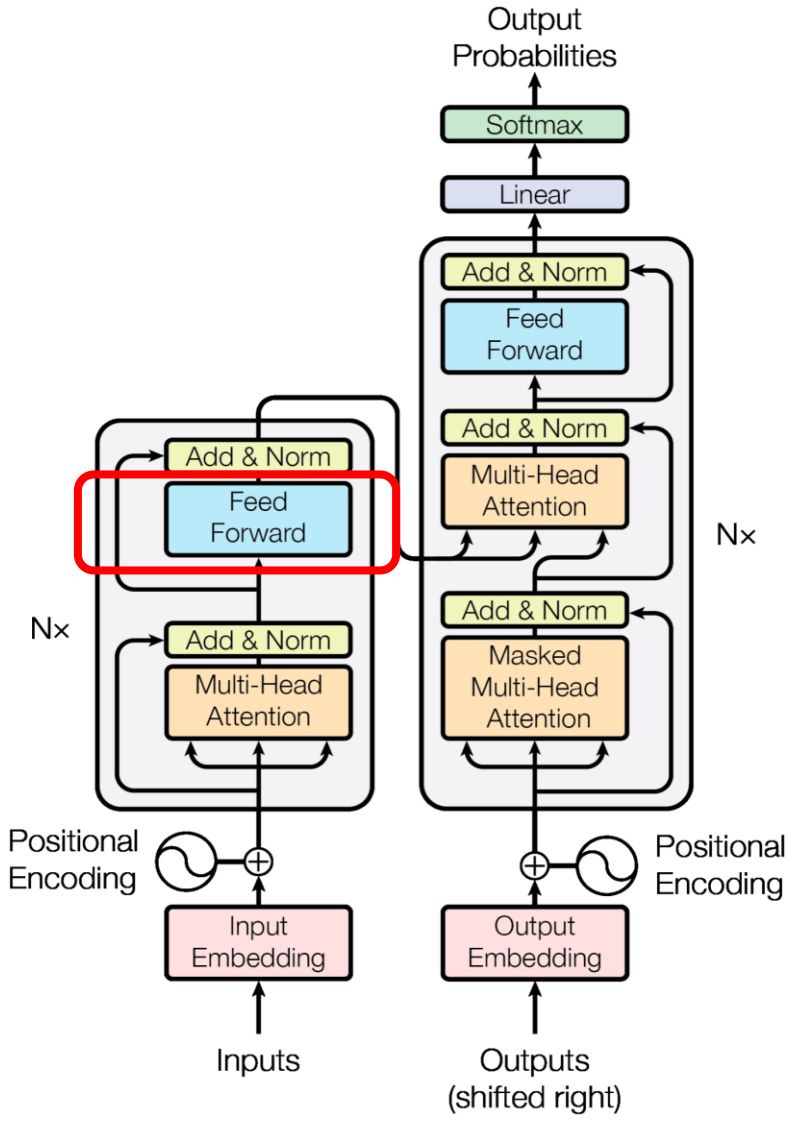
Transformers

Le FFN agit indépendamment sur chaque token.C'est une petite MLP (réseau dense) qui affine les représentations.

Formule : FFN(x)=ReLU($xW1+b1$)$W2+b2$

Rôle :

- Enrichir la représentation du token après qu'il ait intégré le contexte via l'attention.
- Introduire non-linéarité (via ReLU) permet au modèle d'apprendre des transformations complexes.



Transformers

Chaque sous-bloc (attention ou FFN) est entouré par :

- Une connexion résiduelle (“Add”)
- Une normalisation de couche (“Norm”)

Formule: Output=LayerNorm(x+Sublayer(x))

Rôle:

- **Add (connexion résiduelle):**
  - Empêche la perte d’information d’origine.
  - Facilite l’entraînement en évitant le vanishing gradient.
  - Permet au réseau d’apprendre des ajustements plutôt que de tout reconstruire.
- **Norm (Layer Normalization) :**
  - Stabilise les activations.
  - Accélère la convergence.