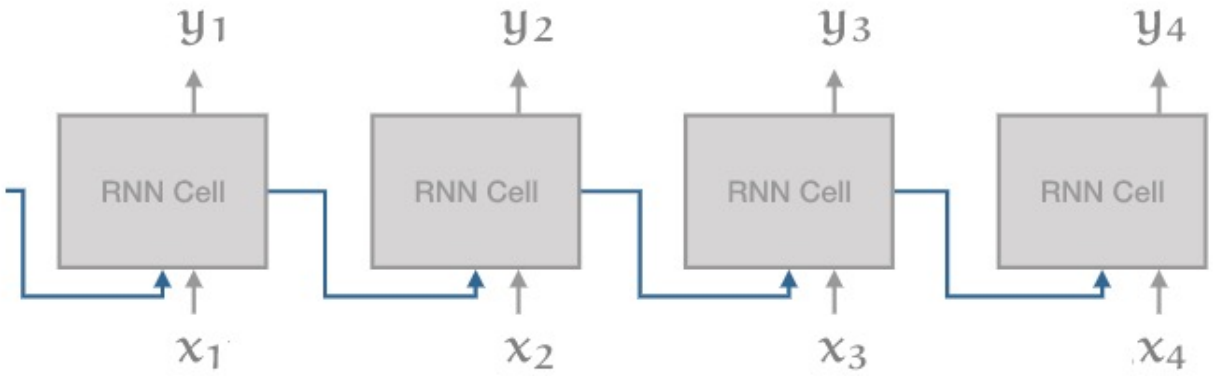
Les Transformers
principes et architecture

Une liste (partiellement ordonnée) de sources

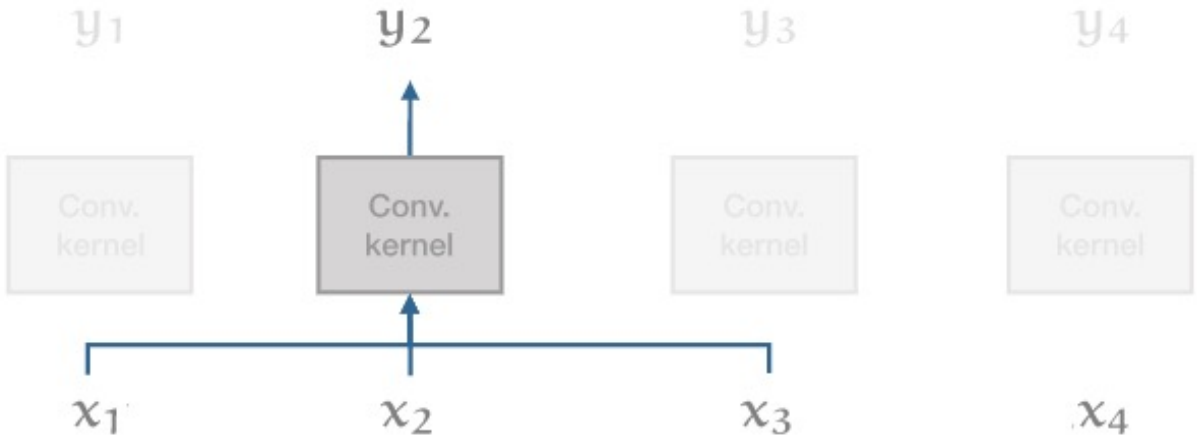
Prérequis
Seq2Seq with Attention and Beam Search

Articles choisis sur le(s) Transformer(s)
1. The Illustrated Transformer (introduction visuelle)
2. Transformers from Scratch (présentation concise + code)
3. Attention is All You Need (article original)
4. The Annotated Transformer (article original + code commenté)
5. BERT – Pre-training of Deep Bidirectional Transformers for Language Understanding (article original)
6. Transformers are Graph Neural Networks (lien avec les GNN)
7. Efficient Transformers: A Survey (variantes pour des textes longs)
8. On the Opportunities and Risks of Foundation Models (analyse critique pluridisciplinaire)

Il était une fois les modèles seq2seq…

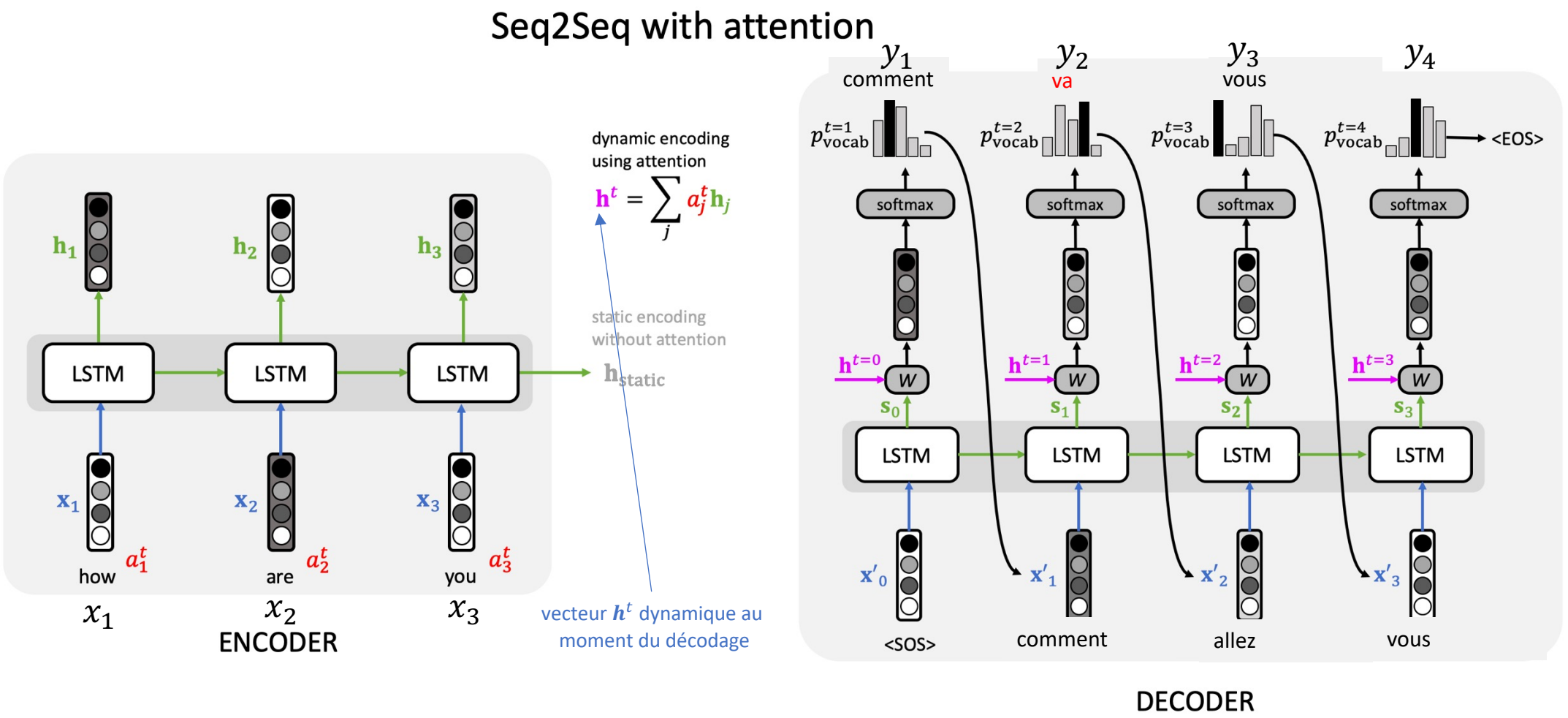


RNN’s
Atout : longueur arbitraire
Inconvénient : non parallélisable



1D – CNN’s
Atout : parallélisable
Inconvénient : longueur limitée

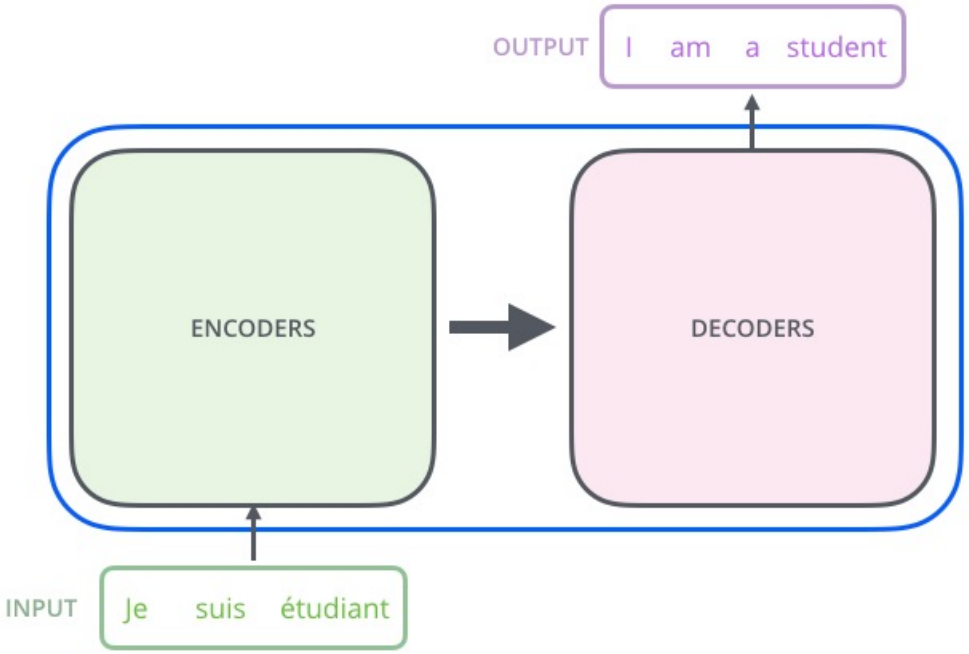
Il était une fois le mécanisme d'attention…

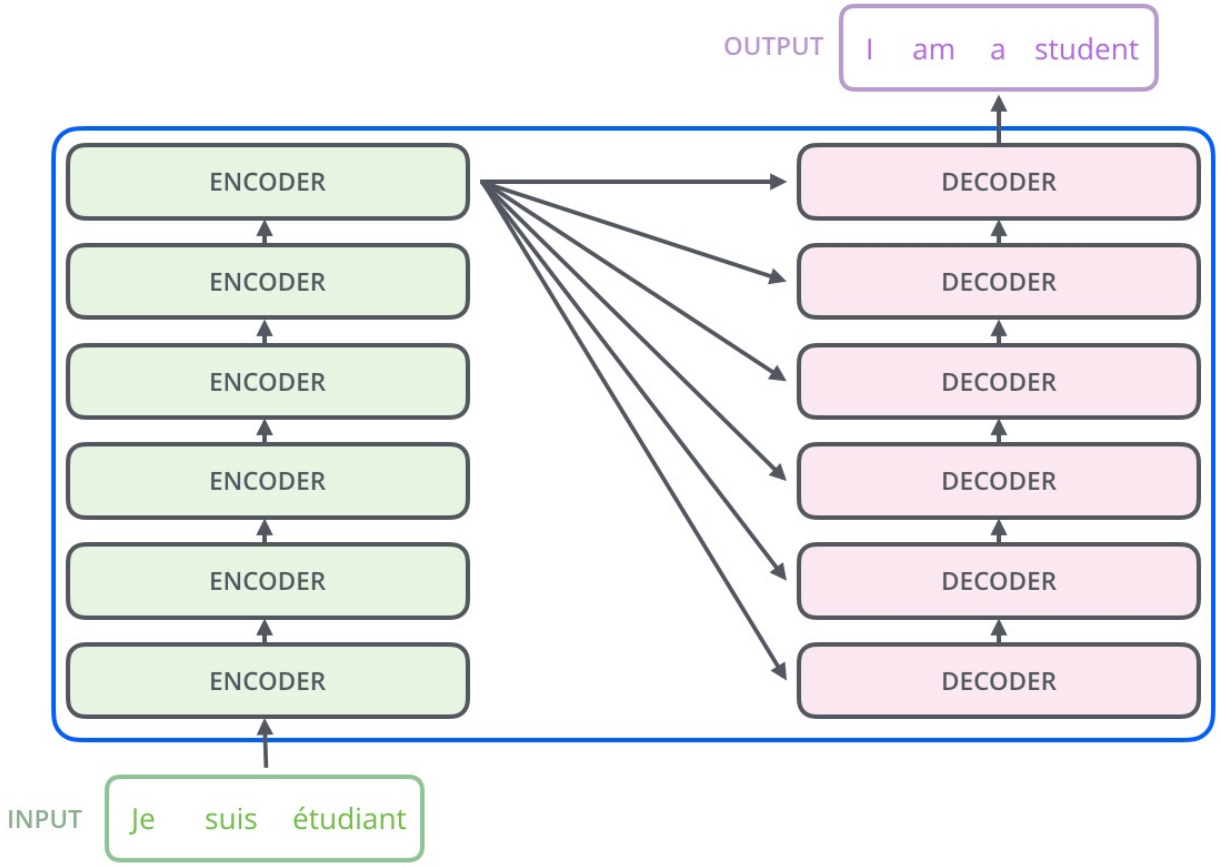


But : modéliser la distribution $P(y_{t+1}|y_1, \ldots, y_t; x_1, \ldots, x_n)$

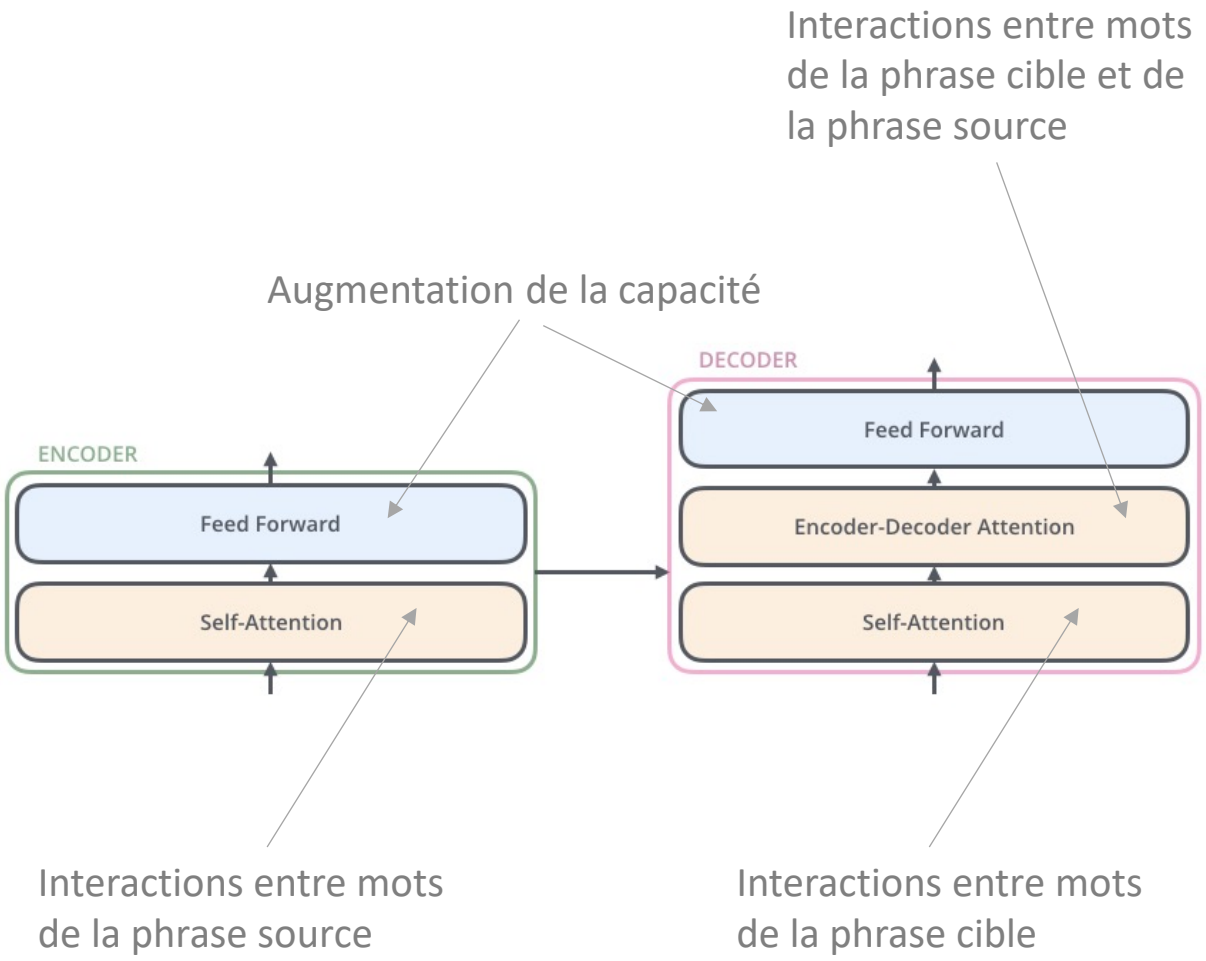
But : concilier les atouts des RNN et des CNN

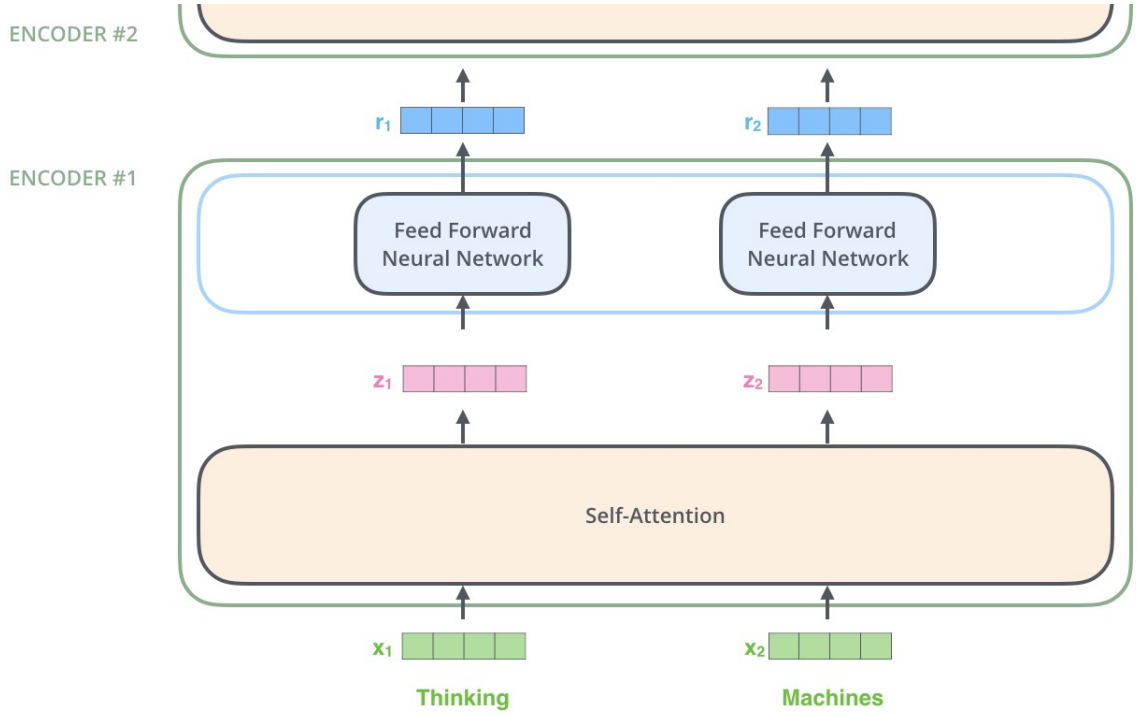
Le Transformer vu comme un ensemble de poupées russes





Ouvrons le Transformer !

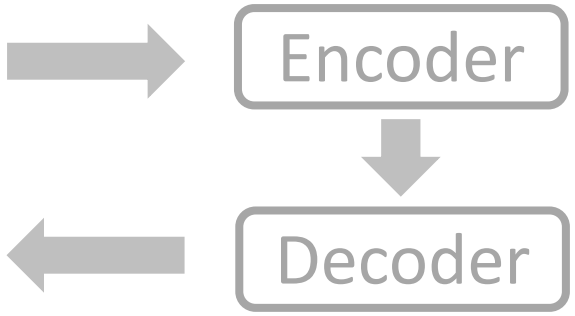


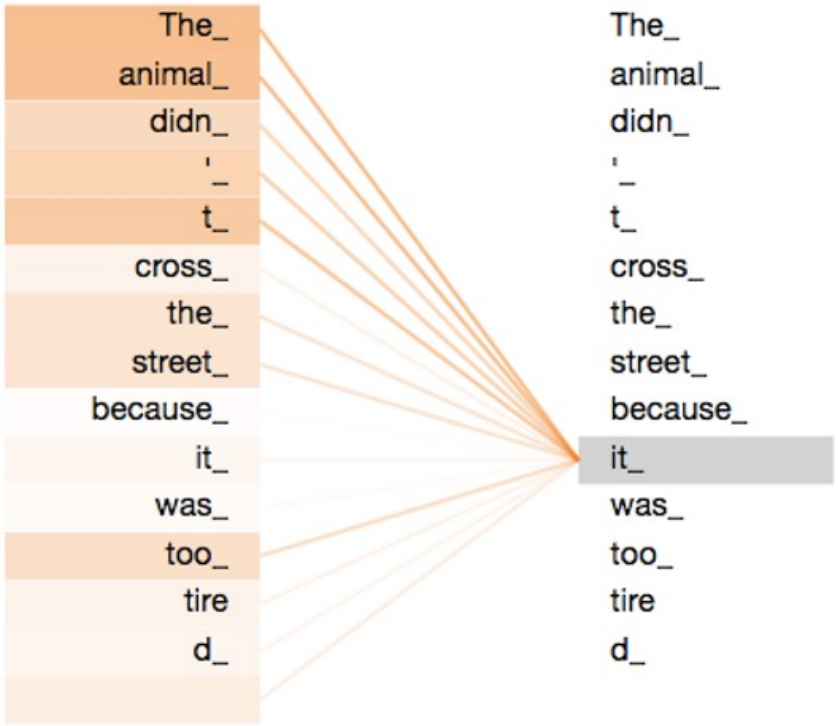


Utilité d'un mécanisme d'attention

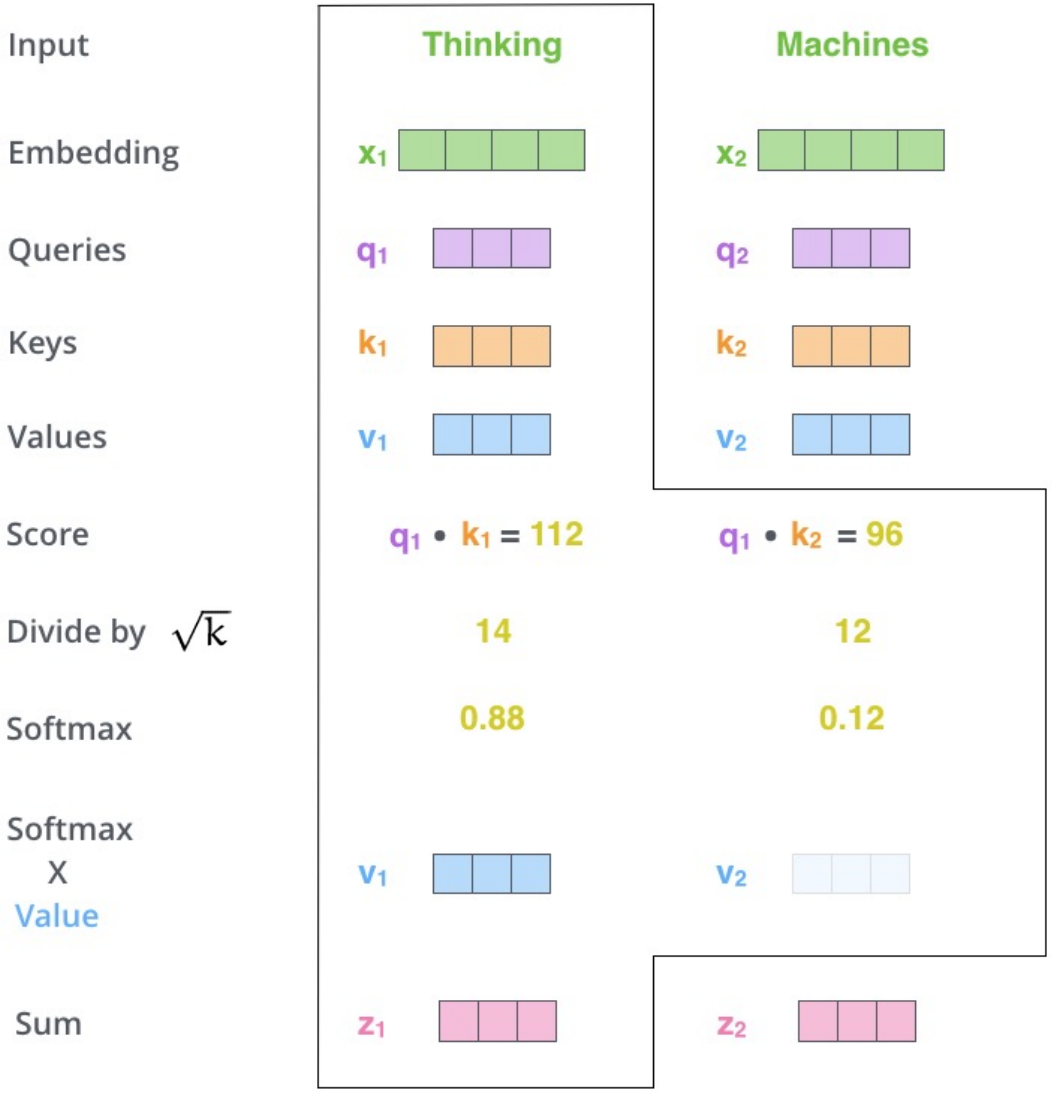
The animal didn't cross the street because it was too wide.
The animal didn't cross the street because it was too tired.

L'animal n'a pas traversé la rue car elle était trop large.
L'animal n'a pas traversé la rue car il était trop fatigué.





Implémentation du MA : key, query, value



On associe 3 vecteurs dans $\mathbb{R}^k$ à chaque vecteur $\mathbf{x}_i$:

- Un vecteur « value » $\mathbf{v}_i$: contribution
- Un vecteur « key » $\mathbf{k}_i$: identification
- Un vecteur « query » $\mathbf{q}_i$: recherche

3 matrices à apprendre

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i \qquad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i \qquad \mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i$$

$$w'_{ij} = \frac{\mathbf{q}_i^{\top} \mathbf{k}_j}{\sqrt{k}} \qquad i, j = 1, \dots L$$

$$w_{ij} = \mathrm{softmax}(w'_{ij})$$

$$\mathbf{z}_i = \sum_j w_{ij} \mathbf{v}_j \, .$$

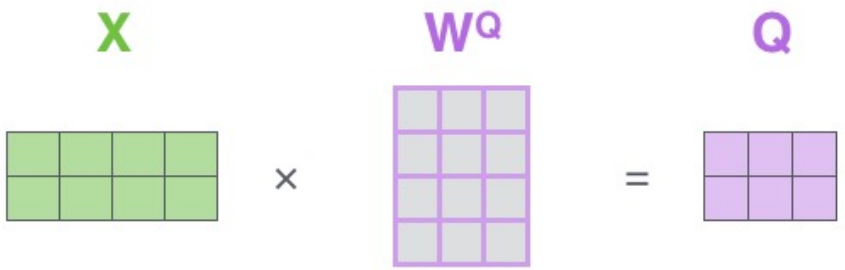
Transformers ⚠ $O(k\,L^2)$

RNN $O(k^2 L)$

$\Rightarrow L < k$

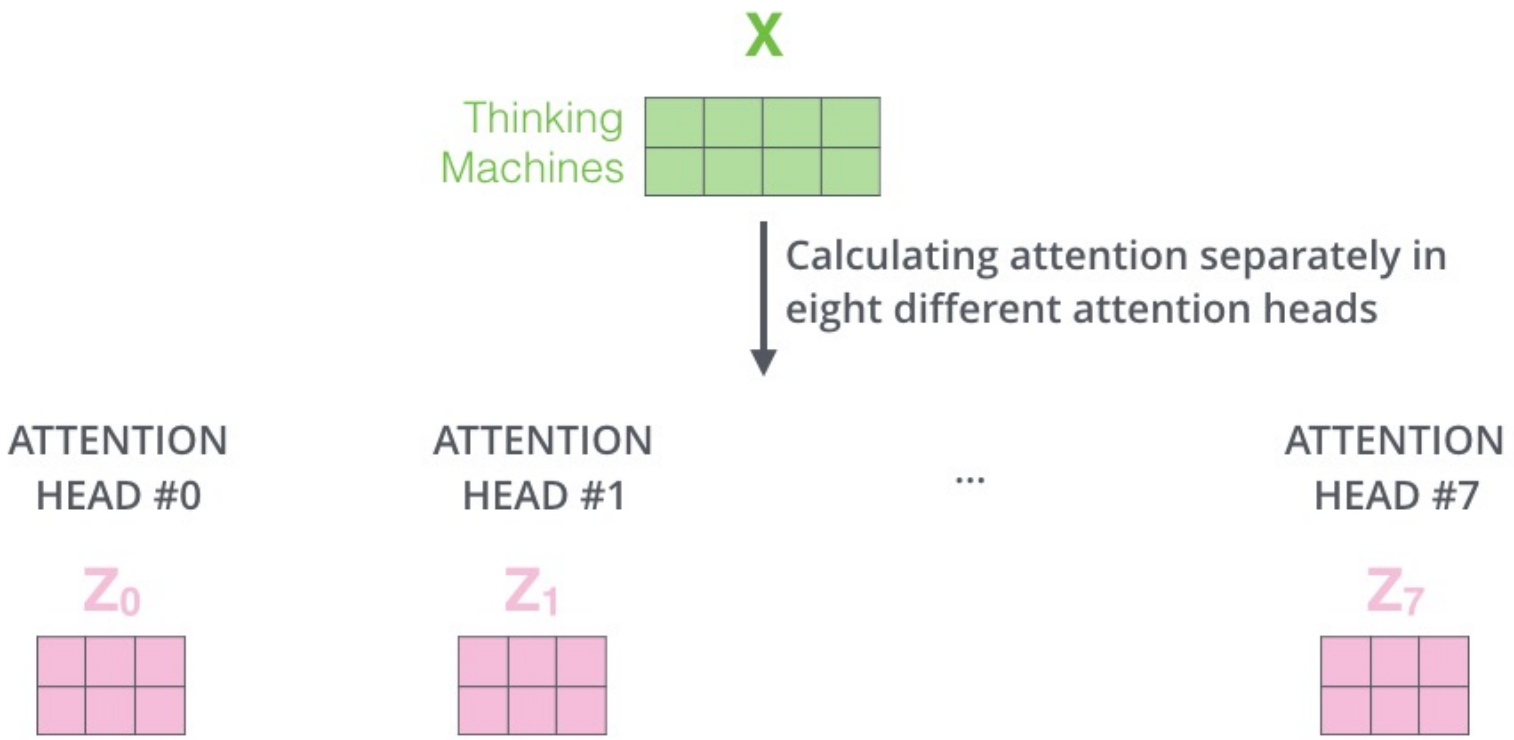
Formulation matricielle du calcul de l'attention

Chaque ligne correspond à un mot en entrée



$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

Plusieurs têtes !



Deux interprétations possibles du « Multi-head Attention Mechanism » :

1. Identifier plusieurs types de relations.

2. Régulariser le mécanisme d'attention.

On réconcilie tout le monde !

1) This is our input sentence*

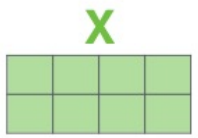
2) We embed each word*

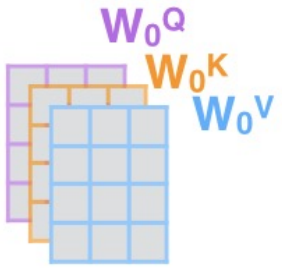
3) Split into 8 heads. We multiply X or R with weight matrices

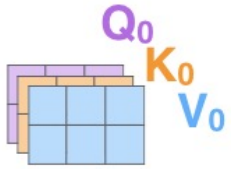
4) Calculate attention using the resulting Q/K/V matrices

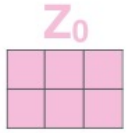
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

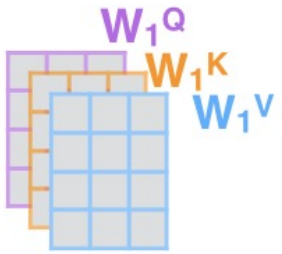
Thinking Machines

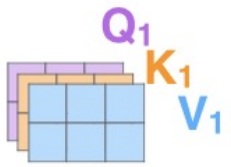


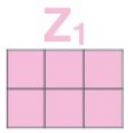


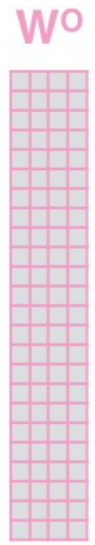


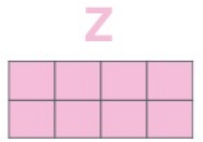




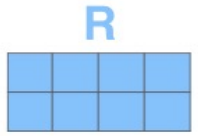


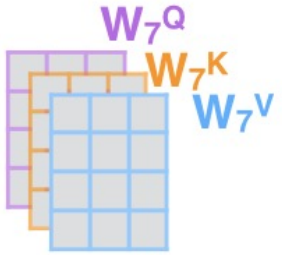


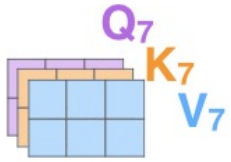


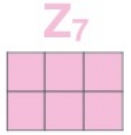


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one





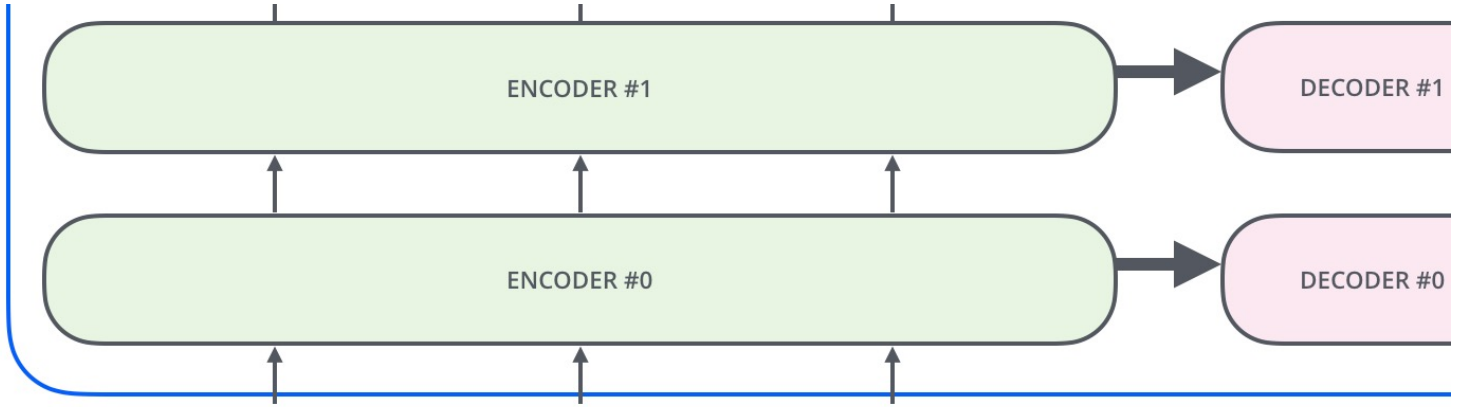


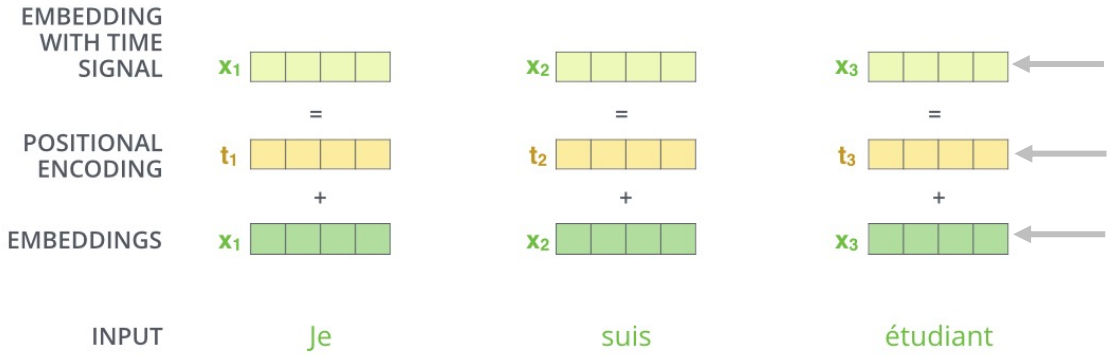


N’oublions pas l’ordre des mots ! ~~bag of words~~

Jusque-là : le Transformer convertit un **ensemble** de vecteurs en un autre **ensemble** de vecteurs.

Ce qui est souhaité : convertir une **liste ordonnée** de vecteurs en une autre **liste ordonnée** de vecteurs.



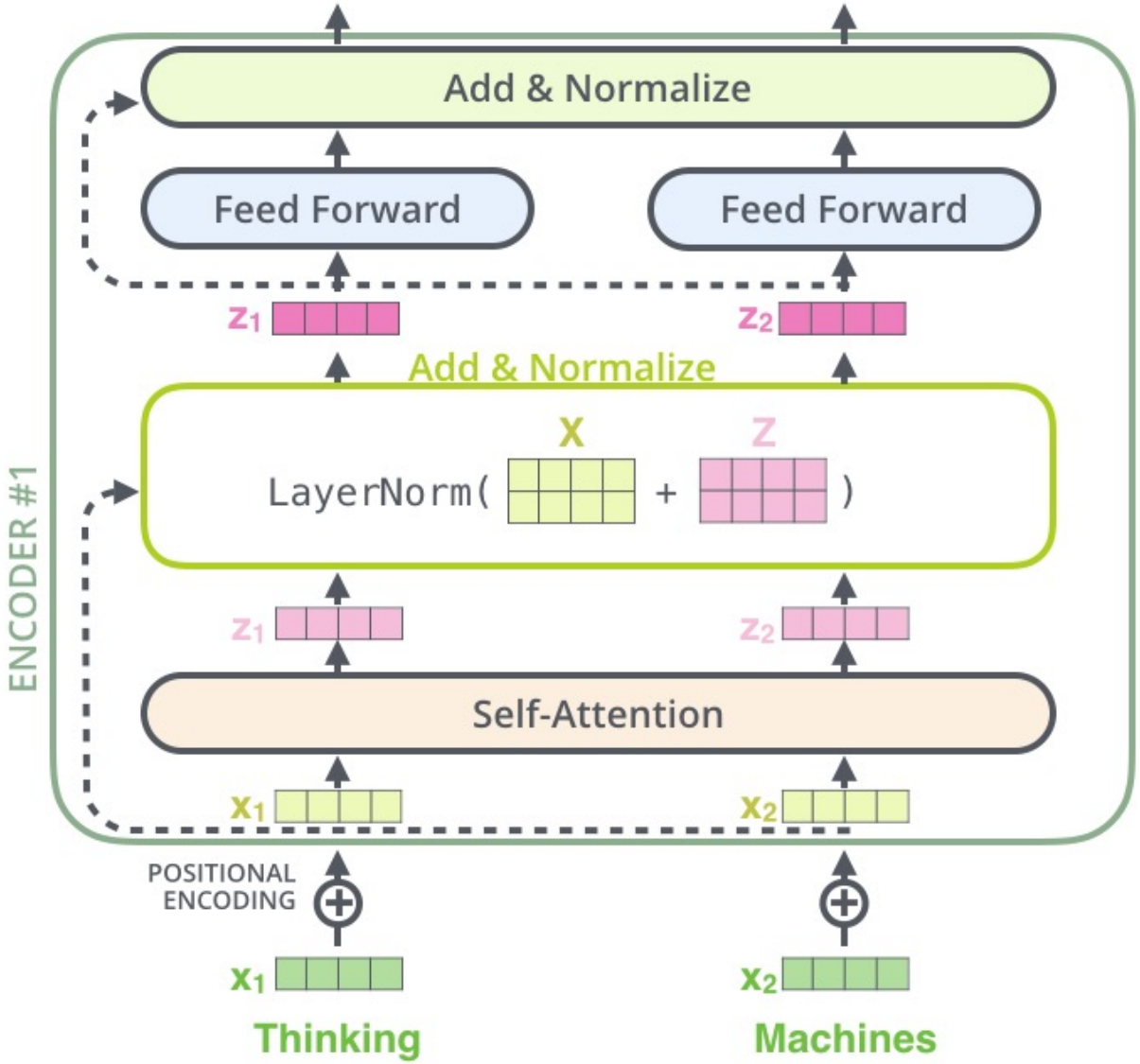


embeddings complétés avec l’information de position

un vecteur $\mathbf{t}_i$ appris ou fixé pour chaque position i

embeddings originaux $\mathbf{x}_i$

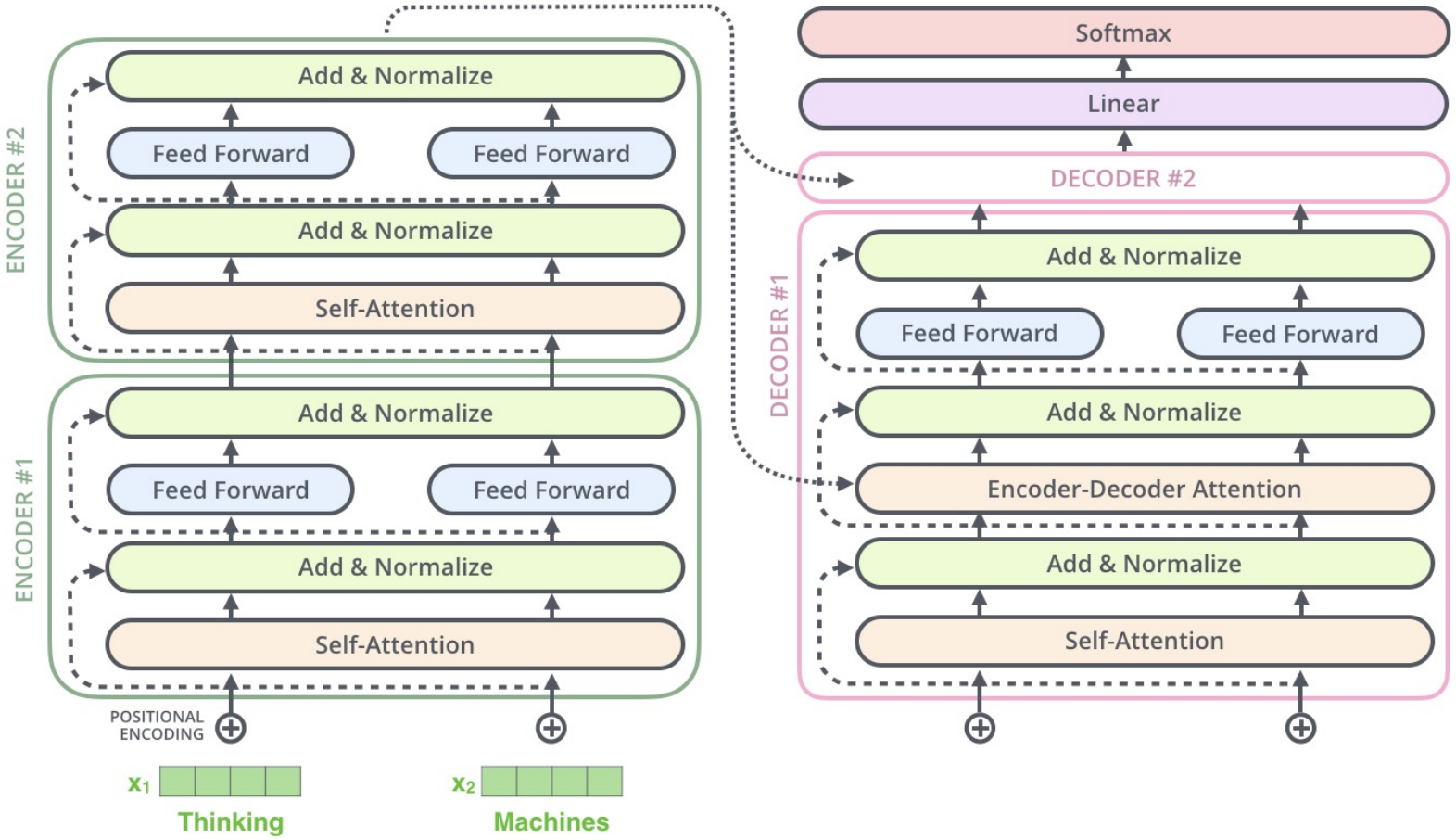
Skip-connections, layer normalization et dropout



Décodeur (1) : un mécanisme d'attention croisé

$\mathbf{k}_1^{(\text{enc}-\text{dec})}, \ldots, \mathbf{k}_L^{(\text{enc}-\text{dec})}$

$\mathbf{v}_1^{(\text{enc}-\text{dec})}, \ldots, \mathbf{v}_L^{(\text{enc}-\text{dec})}$

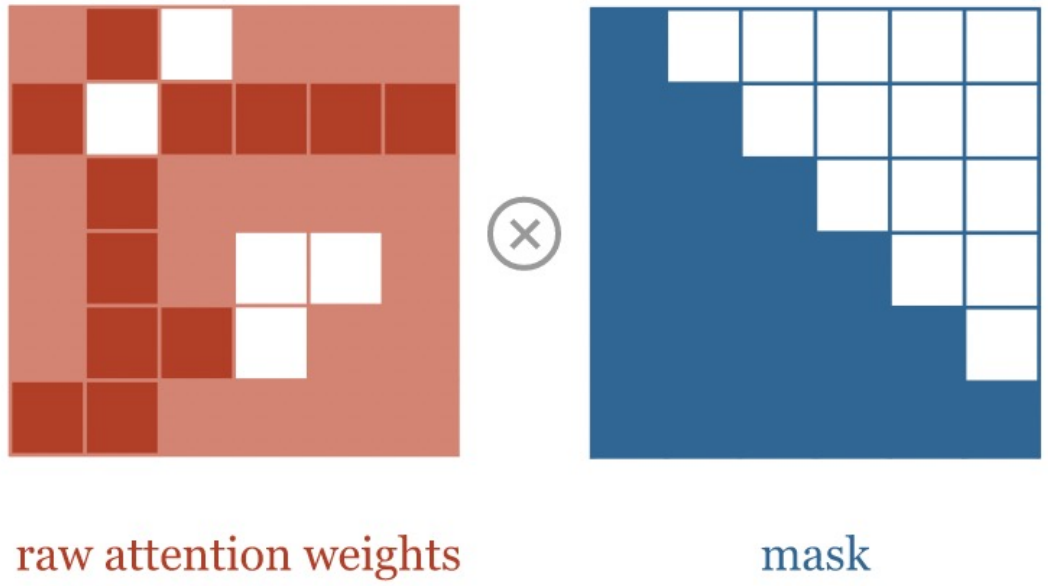


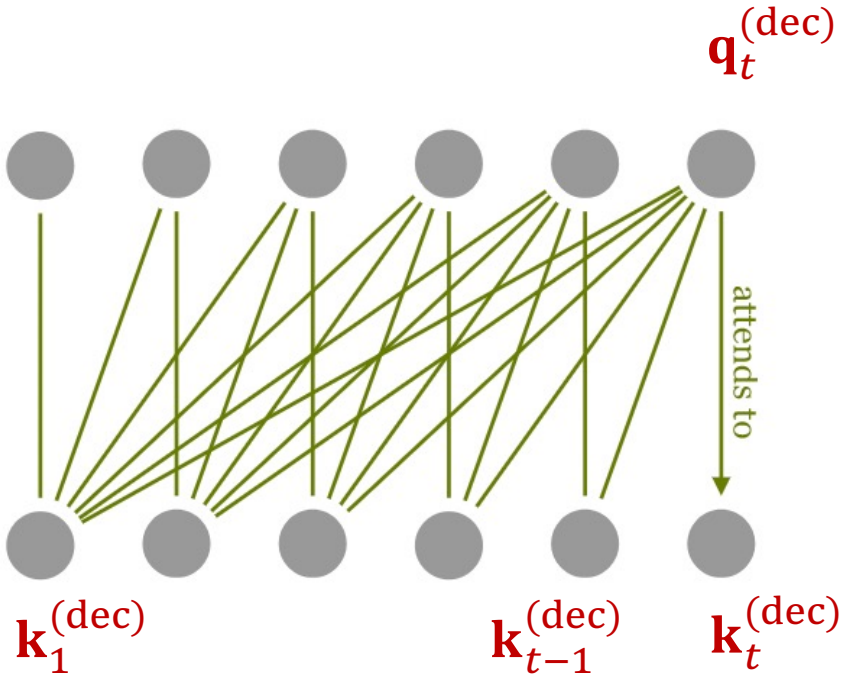
$\mathbf{q}_1^{(\text{enc}-\text{dec}\ \#2)}, \ldots, \mathbf{q}_L^{(\text{enc}-\text{dec}\ \#2)}$

$\mathbf{q}_1^{(\text{enc}-\text{dec}\ \#1)}, \ldots, \mathbf{q}_L^{(\text{enc}-\text{dec}\ \#1)}$

Décodeur (2) : interdiction de regarder le futur !

$$\left(\mathbf{q}_i^{(\mathrm{dec})} \cdot \mathbf{k}_j^{(\mathrm{dec})}\right)_{i=1,\ldots,L}^{j=1,\ldots,L}$$

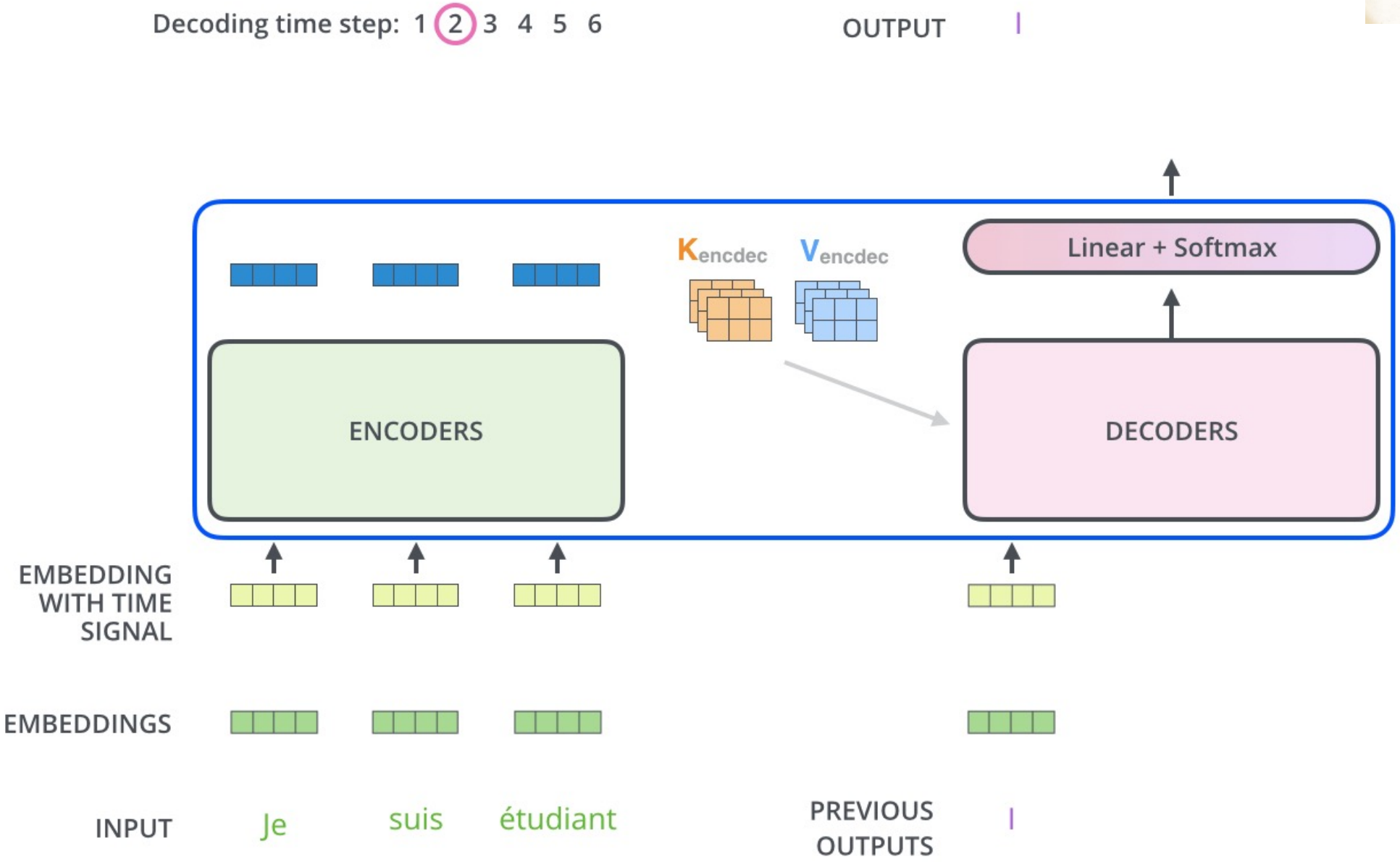




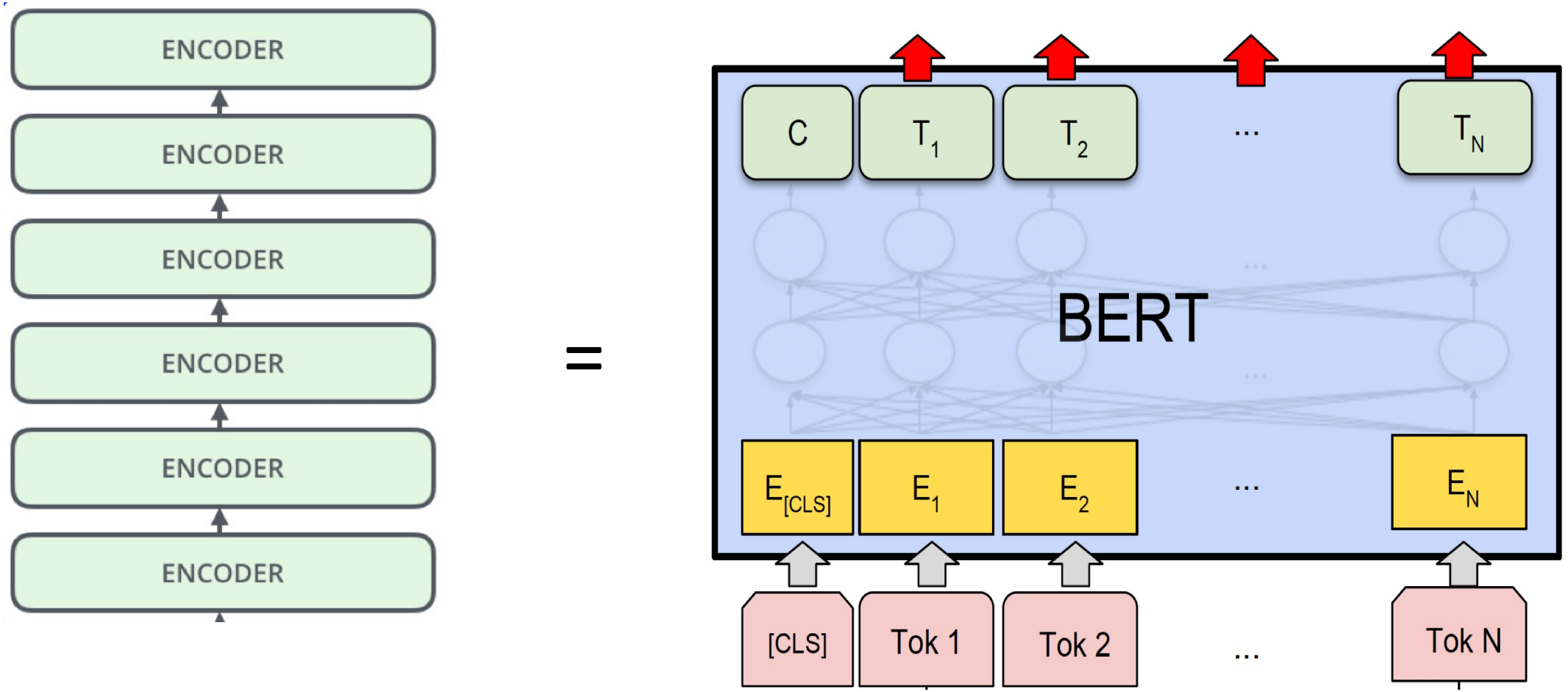
Un **masque** est appliqué avant le softmax du mécanisme de **self-attention** pour interdire au modèle de regarder le futur

Lorsque le décodeur fait une prédiction à l'instant t il n'a accès qu'aux informations antérieures à cet instant. Lors de l'entraînement il faut donc lui **cacher le futur** de t bien qu'il soit connu dans le train set !

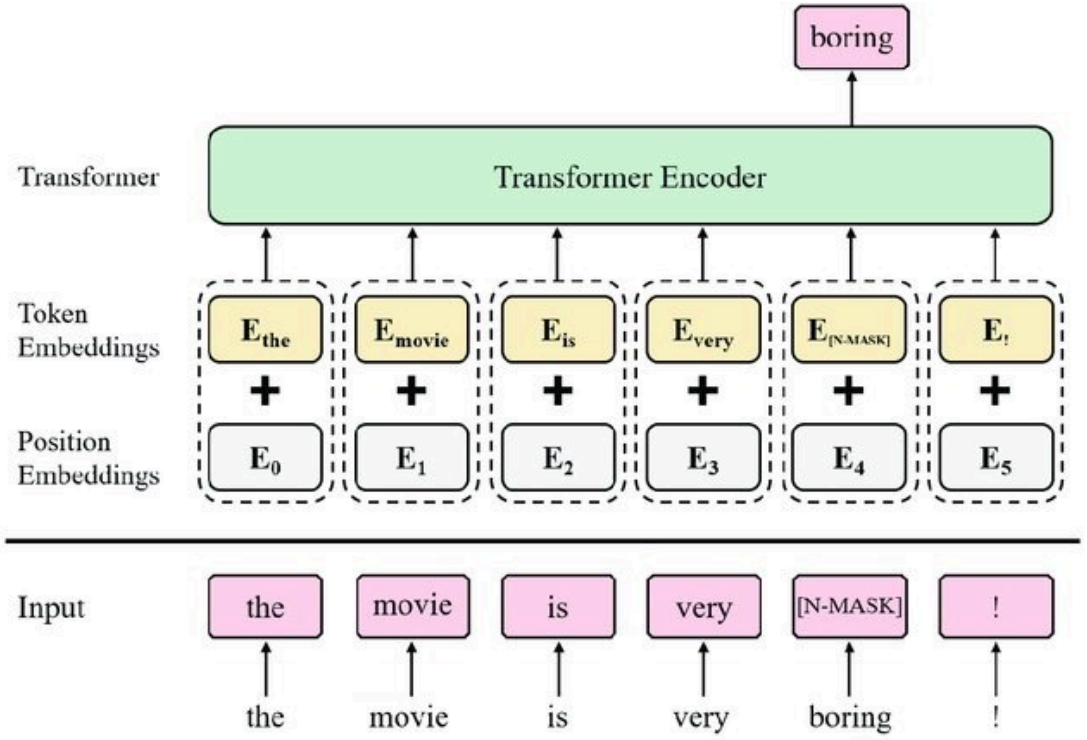
Décodeur (3) : prédiction des mots



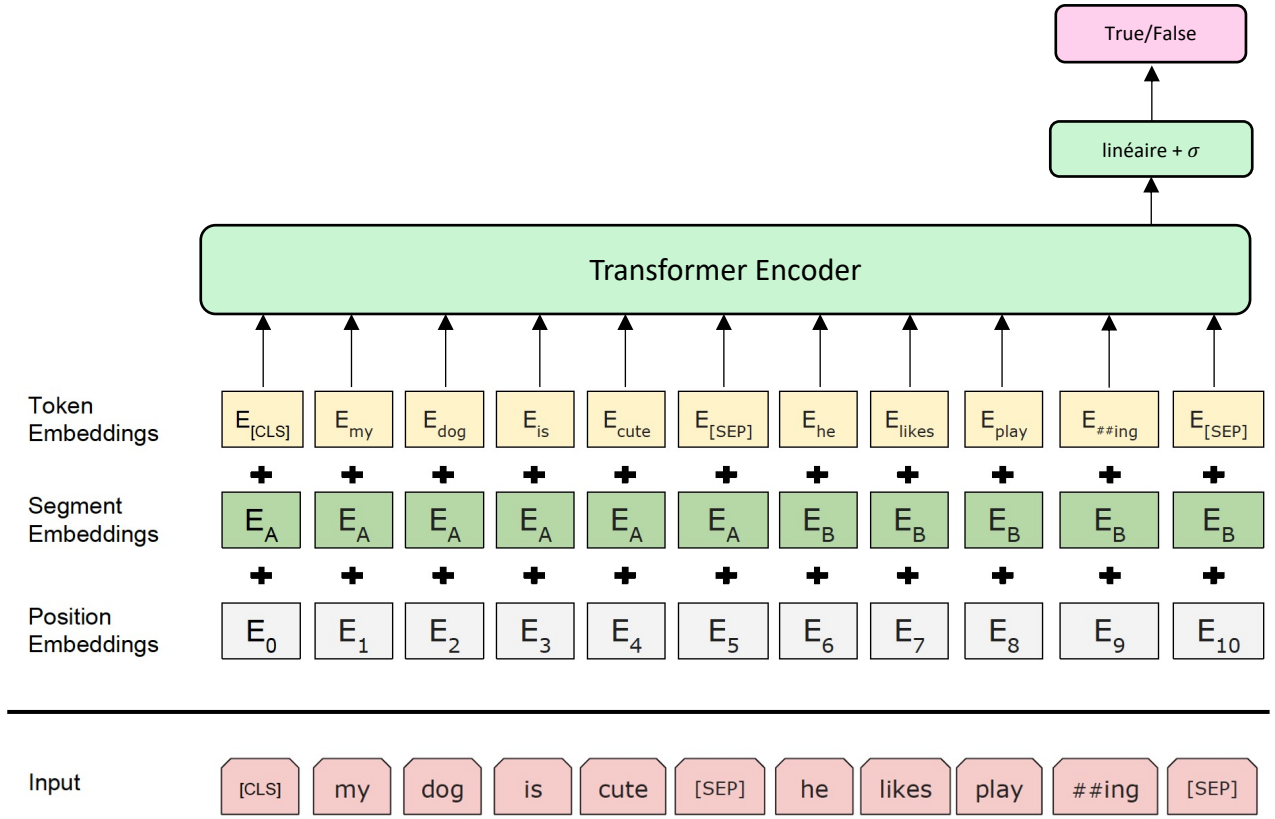
BERT = Encoder Only Transformer



BERT : 2 tâches de pré-entraînement

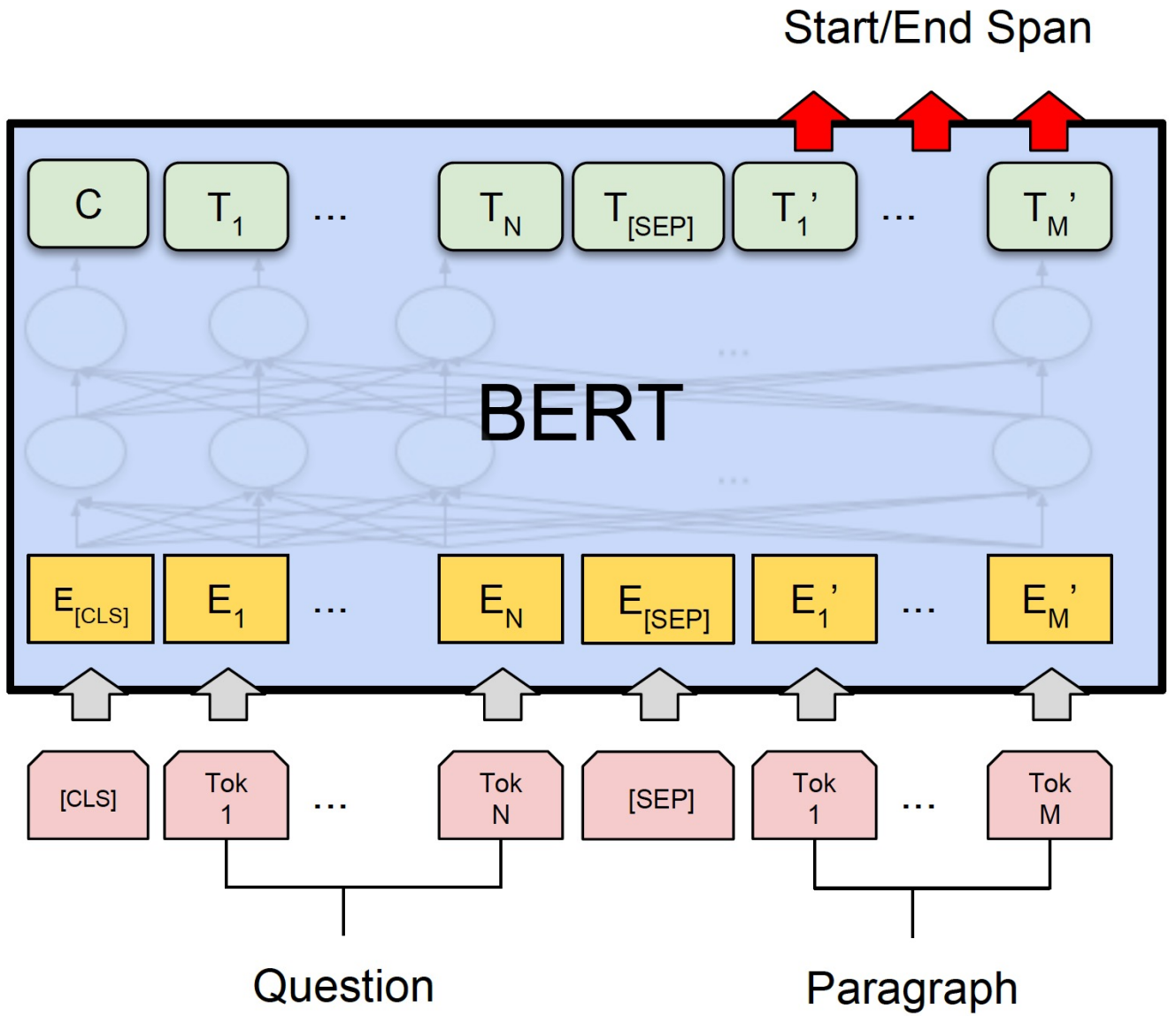


(1) Masked Language Model



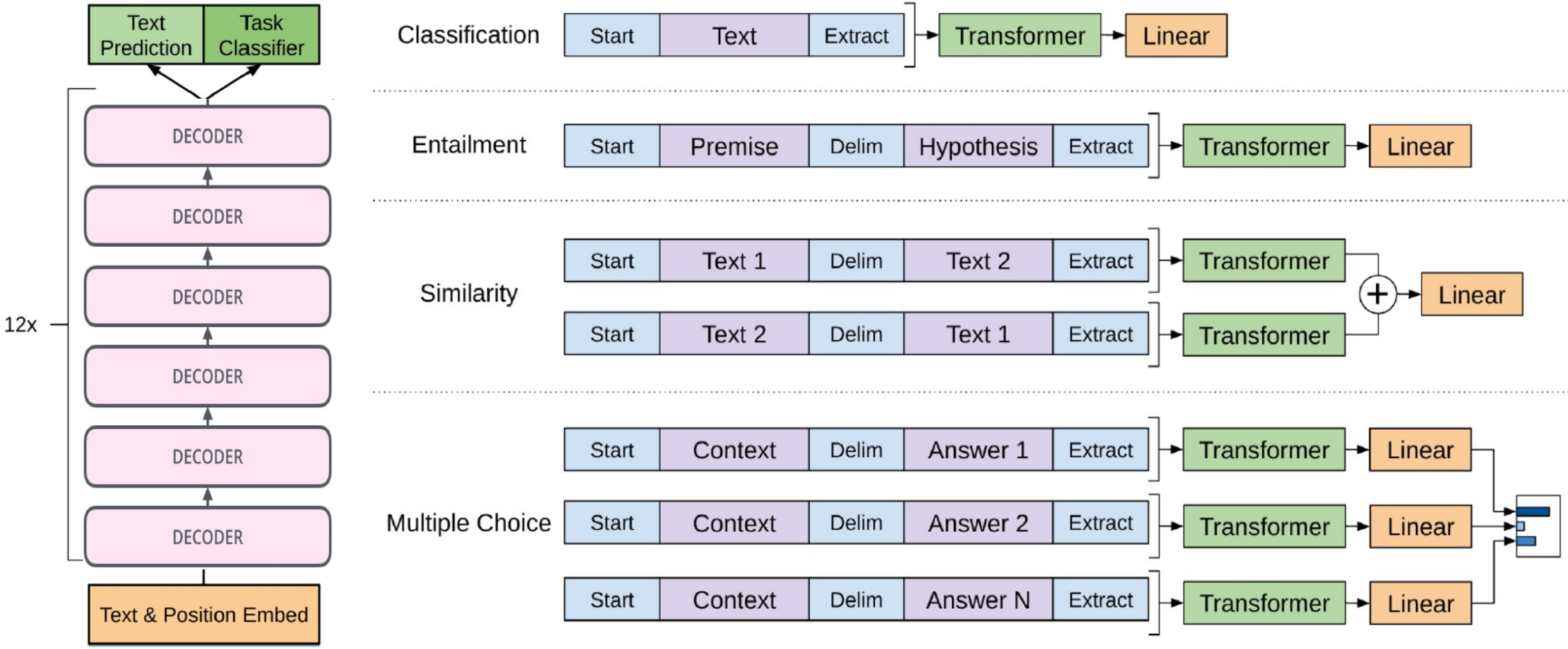
(2) Next Sentence Prediction

BERT : architecture « universelle » pour le NLP



GPT 1 = Decoder Only Transformer + …

pré-entraînement sur LM + adaptation et fine tuning spécifique par tâche

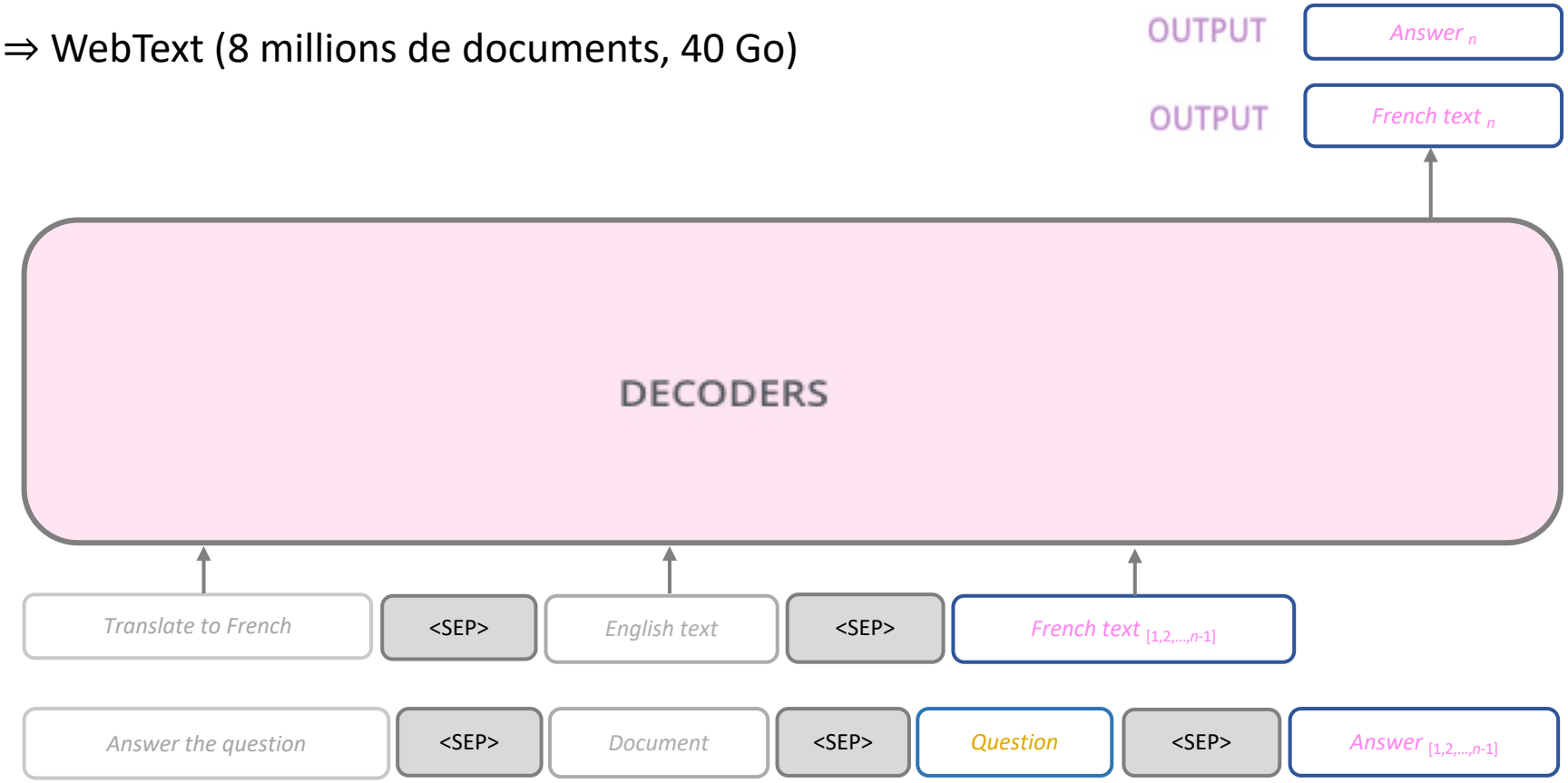


GPT 2 & 3 = Decoder Only Transformer + …

pré-entraînement sur très gros LM + ~~adaptation et fine tuning spécifique par tâche~~

Idée : spécification d'une tâche en langage naturel plutôt qu'en utilisant une architecture dédiée

Data set : Common Crawl ⇒ WebText (8 millions de documents, 40 Go)



…