Architecture de réseau de neurones
-
Le Transformer

Guillaume Bourmaud

PLAN

I. RNN vs Transformer

II. Couche d’attention à softmax

III. Équivariance par permutation et encodage de la position

IV. Application à des images

V. Limites

I) RNN vs Transformer

“Attention is All You Need”, NIPS 2017

I)

Sequence-to-sequence


```
"the cat sat on the mat" -> [Seq2Seq model] -> "le chat etait assis sur le tapis"
```

I)

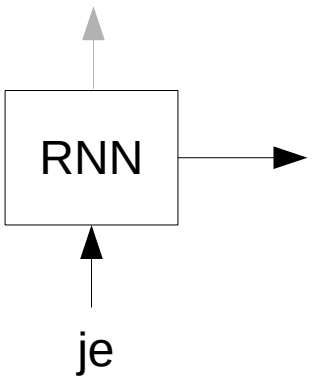
RNN - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”

I)

RNN - inférence

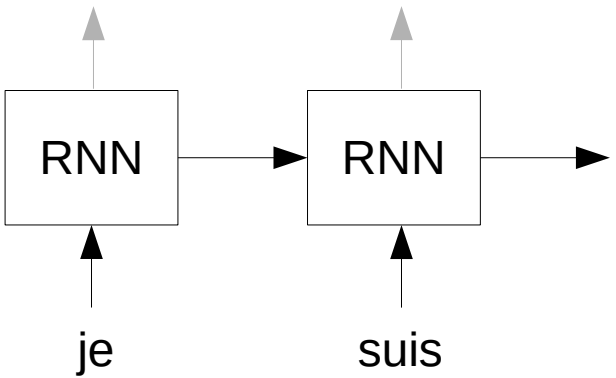
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



I)

RNN - inférence

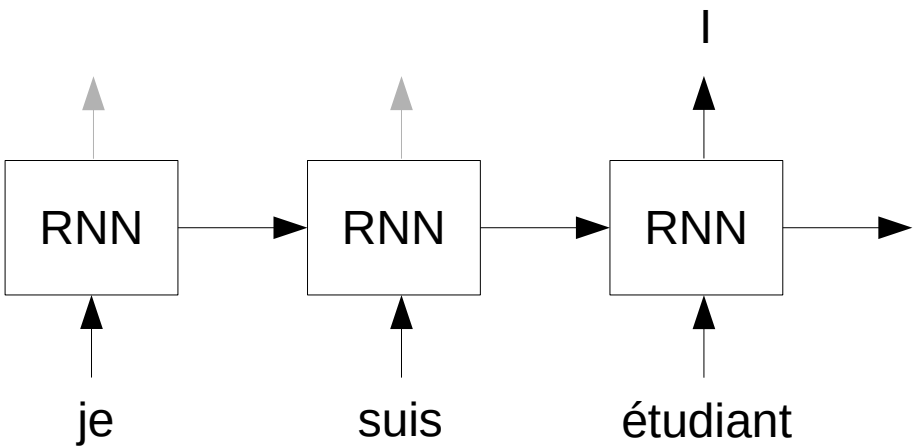
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



I)

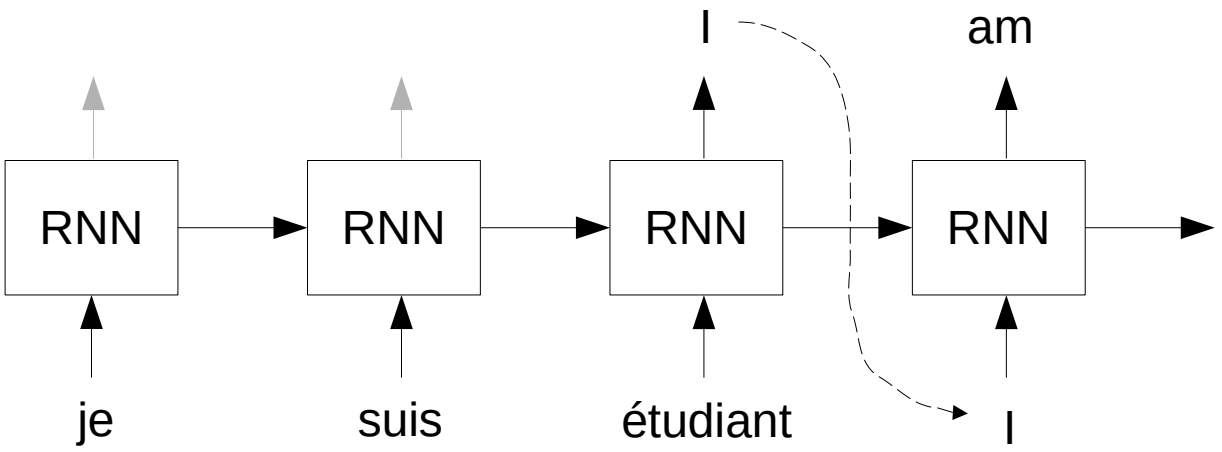
RNN - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



RNN - inférence

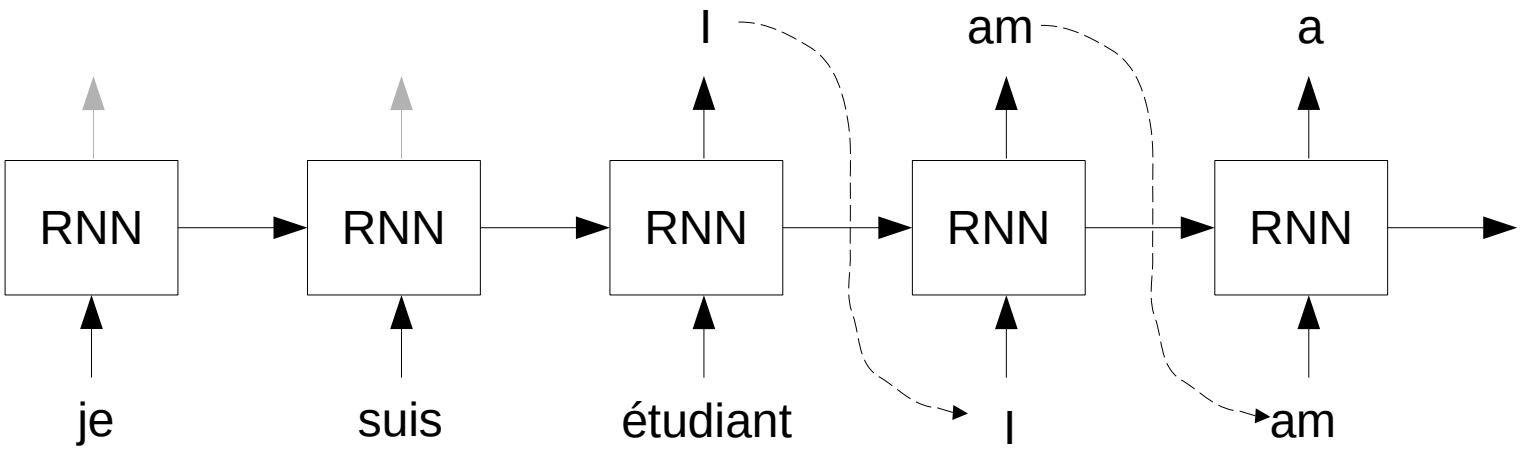
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



I)

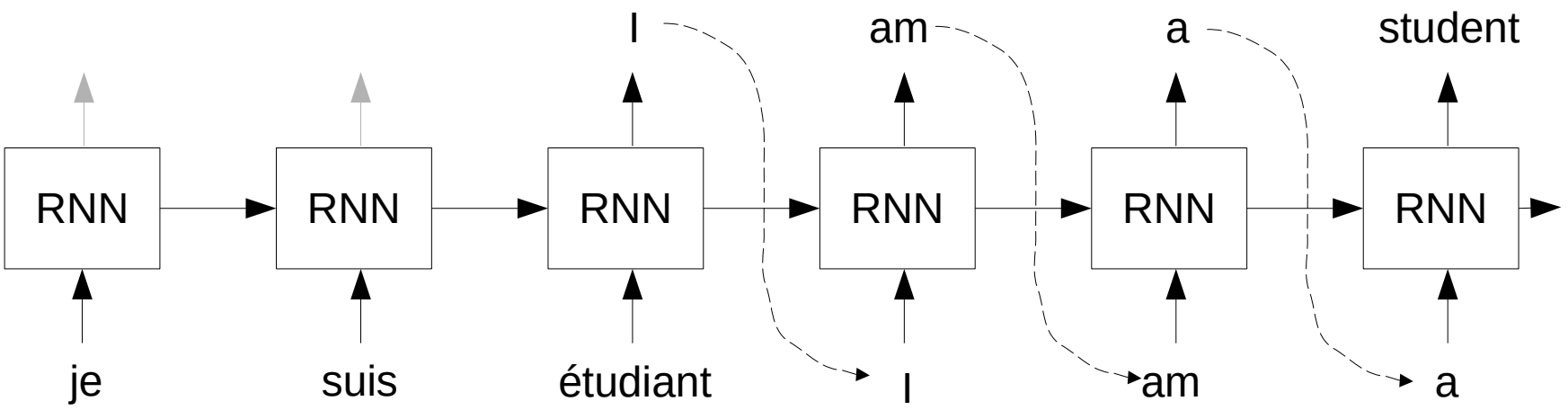
RNN - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



RNN - inférence

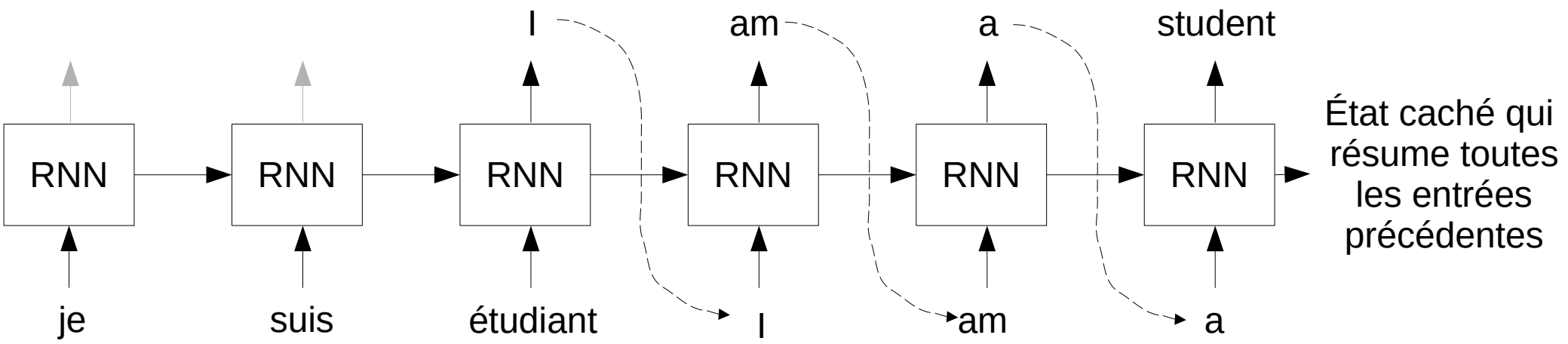
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



I)

RNN - inférence

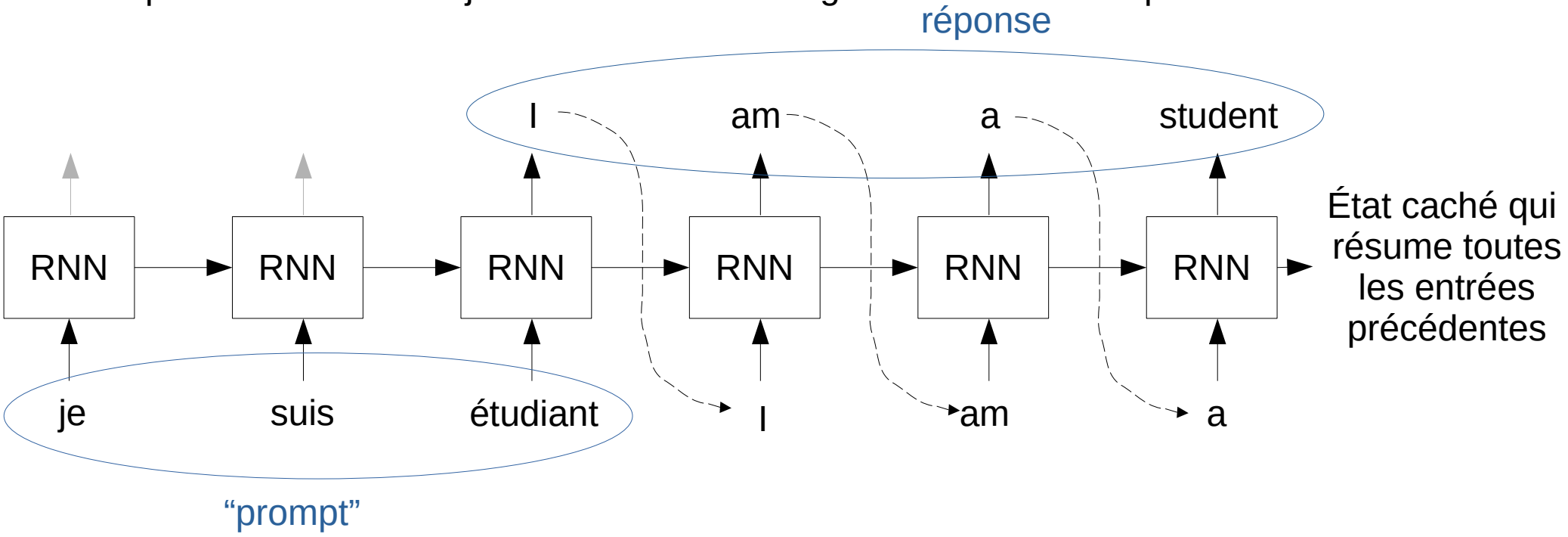
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



I)

RNN - inférence

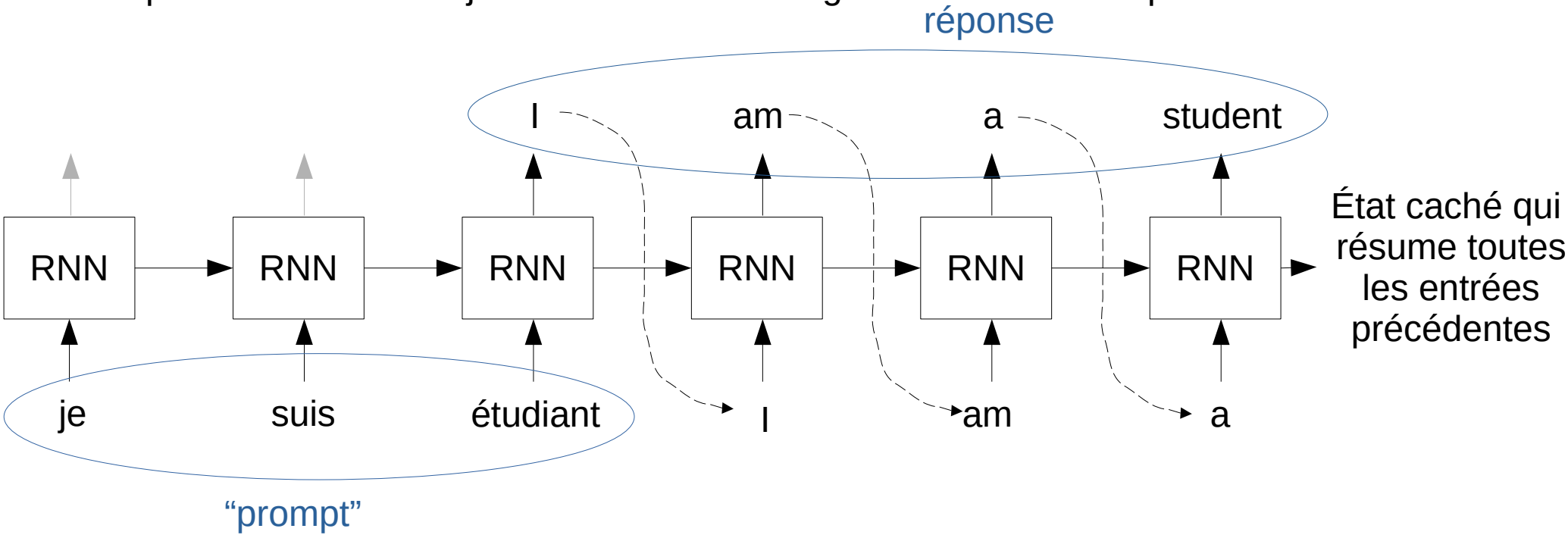
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



I)

RNN - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”

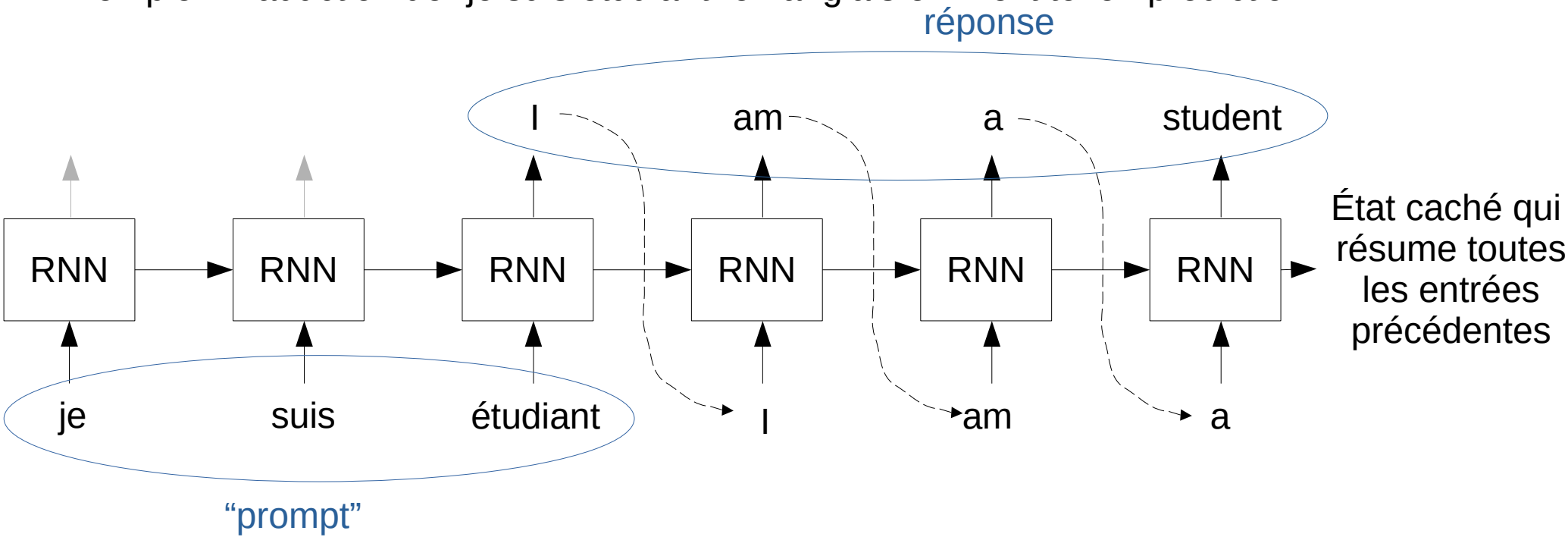


Inférence séquentielle (à cause du paradigme “next token prediction” pas de l’architecture RNN)

I)

RNN - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”

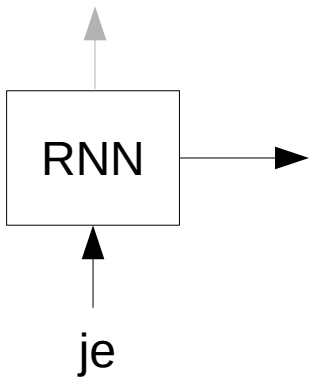


Inférence séquentielle (à cause du paradigme “next token prediction” pas de l’architecture RNN)

→ L’architecture Transformer ne va pas régler ce problème (ChatGPT est séquentiel)

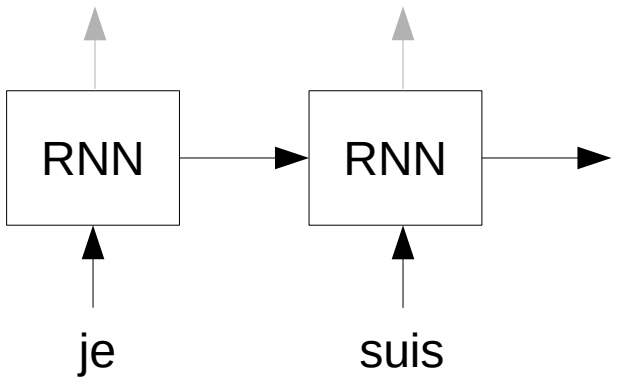
RNN - apprentissage

“Teacher forcing”



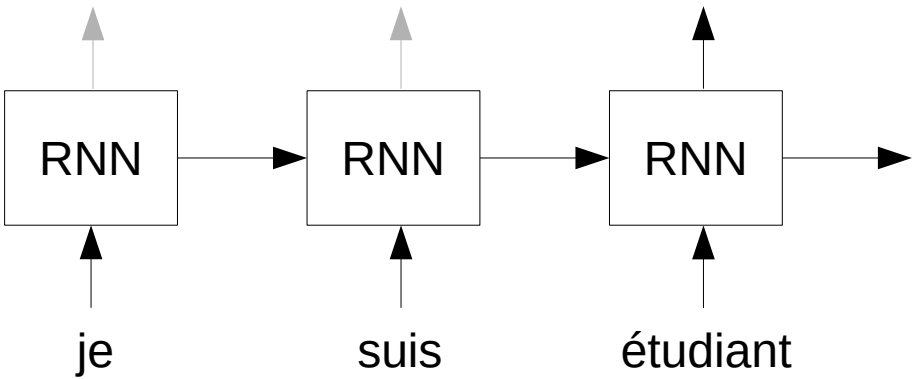
RNN - apprentissage

“Teacher forcing”



RNN - apprentissage

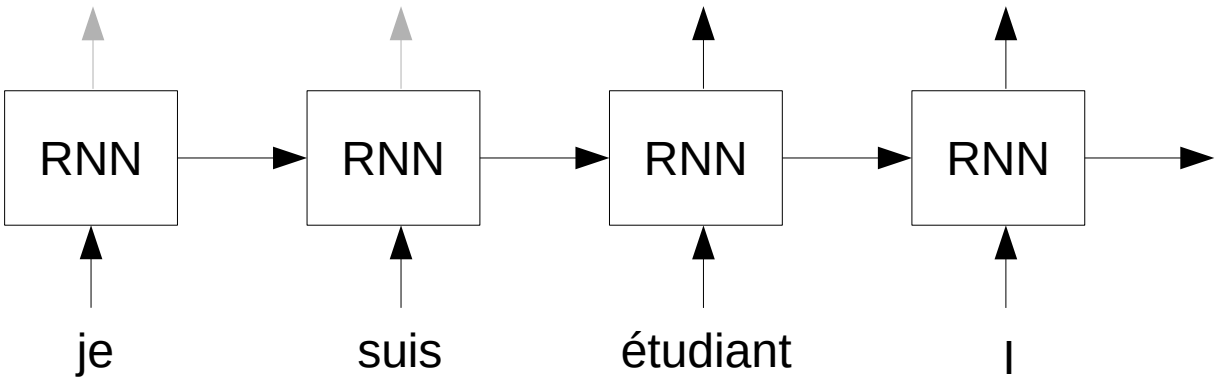
“Teacher forcing”



I)

RNN - apprentissage

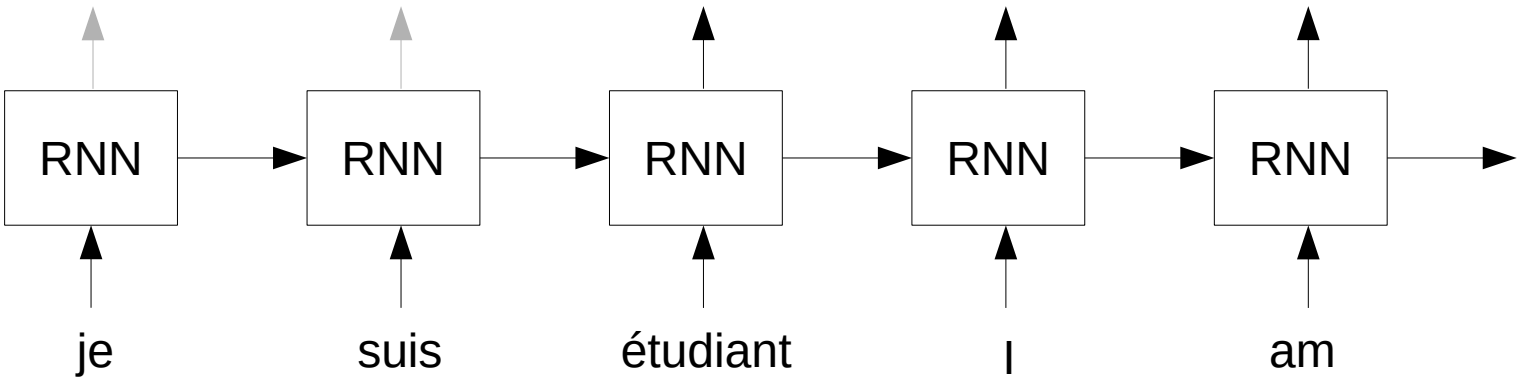
“Teacher forcing”



I)

RNN - apprentissage

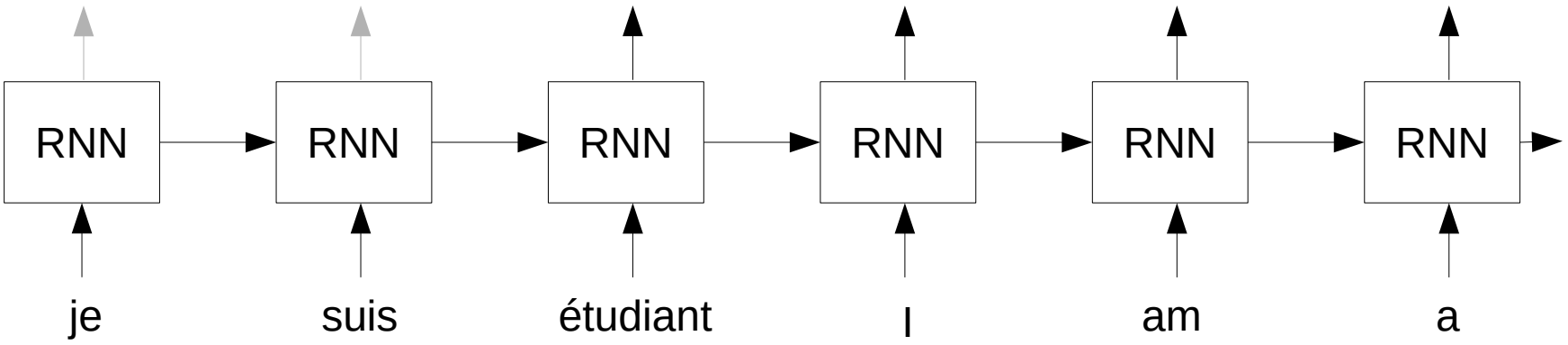
“Teacher forcing”



I)

RNN - apprentissage

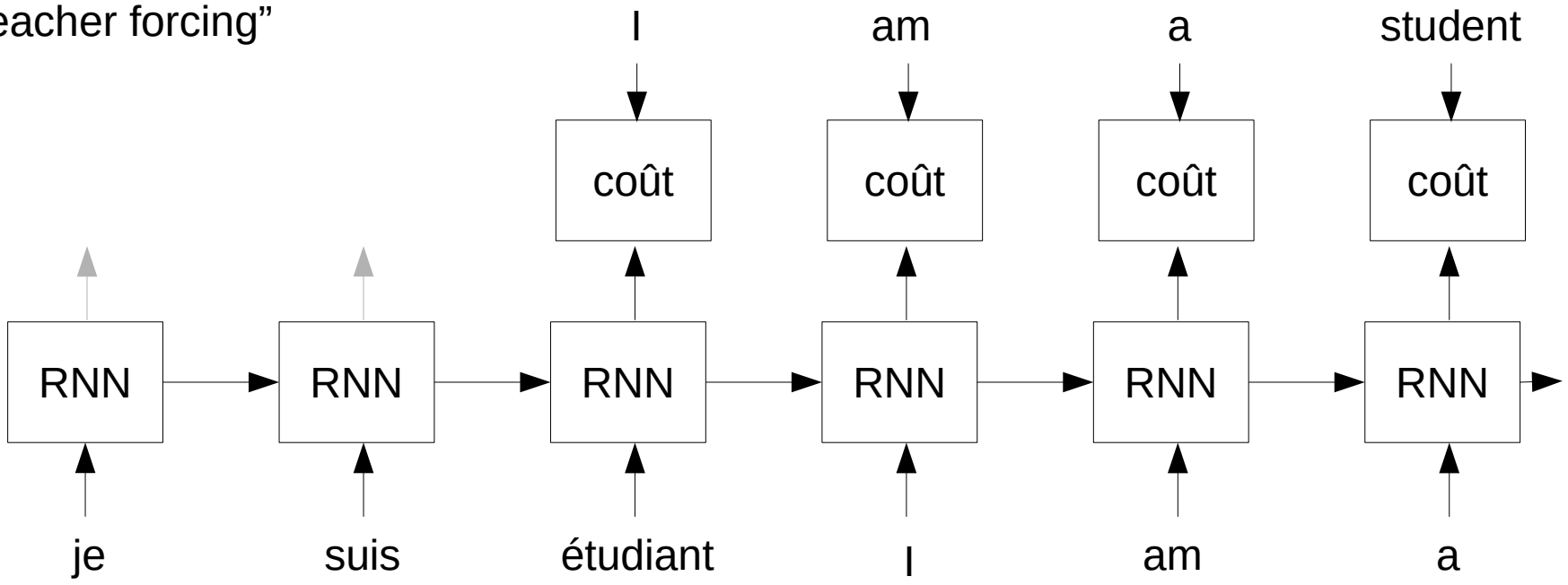
“Teacher forcing”



I)

RNN - apprentissage

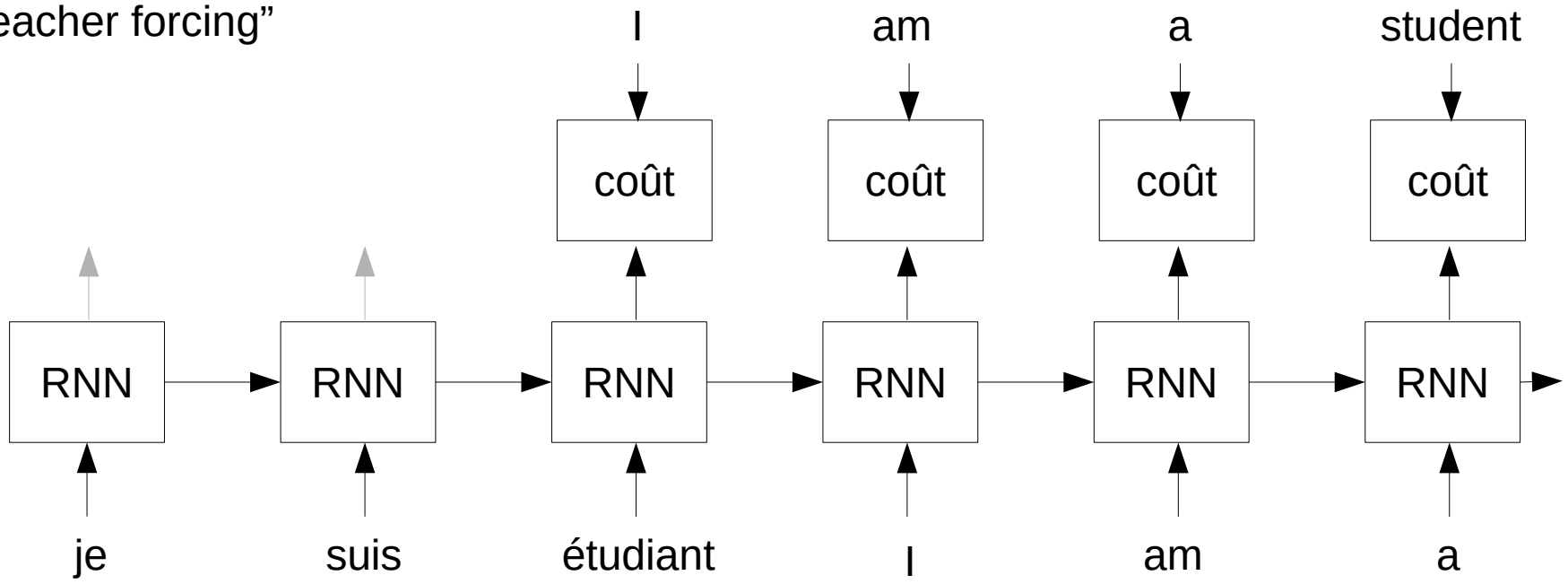
“Teacher forcing”



I)

RNN - apprentissage

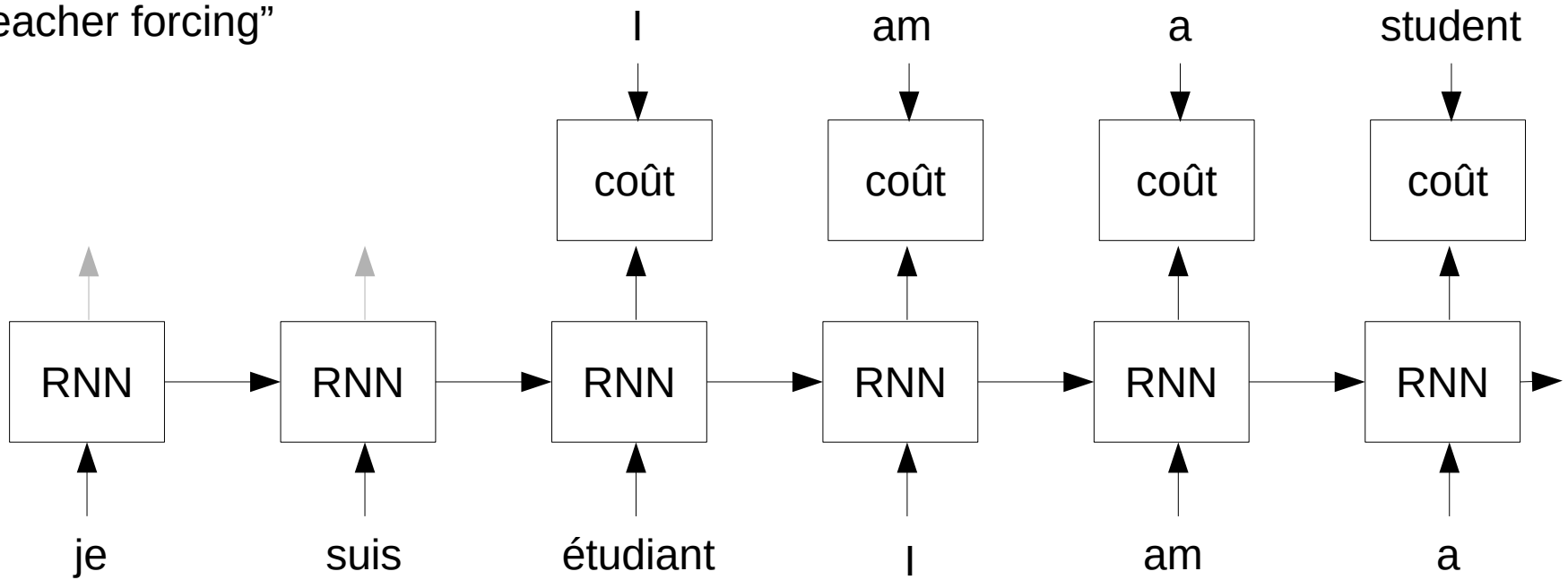
“Teacher forcing”



- Séquentiel (Non parallélisable) car il y a un **état caché** à propager.
→ apprentissage lent (à cause de l’architecture RNN)

RNN - apprentissage

“Teacher forcing”



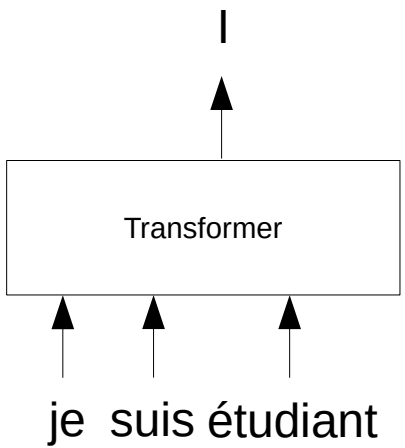
- Séquentiel (Non parallélisable) car il y a un **état caché** à propager.
→ apprentissage lent (à cause de l’architecture RNN)

- Doit apprendre à résumer à chaque instant toutes les entrées précédentes dans l’**état caché**
→ problème des longues dépendances (à cause de l’architecture RNN)

I)

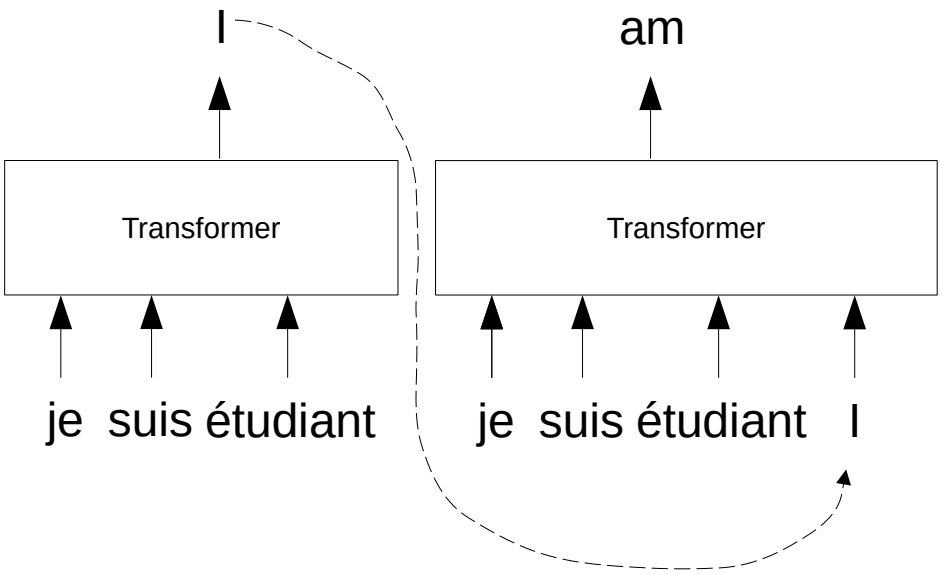
Transformer (“decoder only”) - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



Transformer (“decoder only”) - inférence

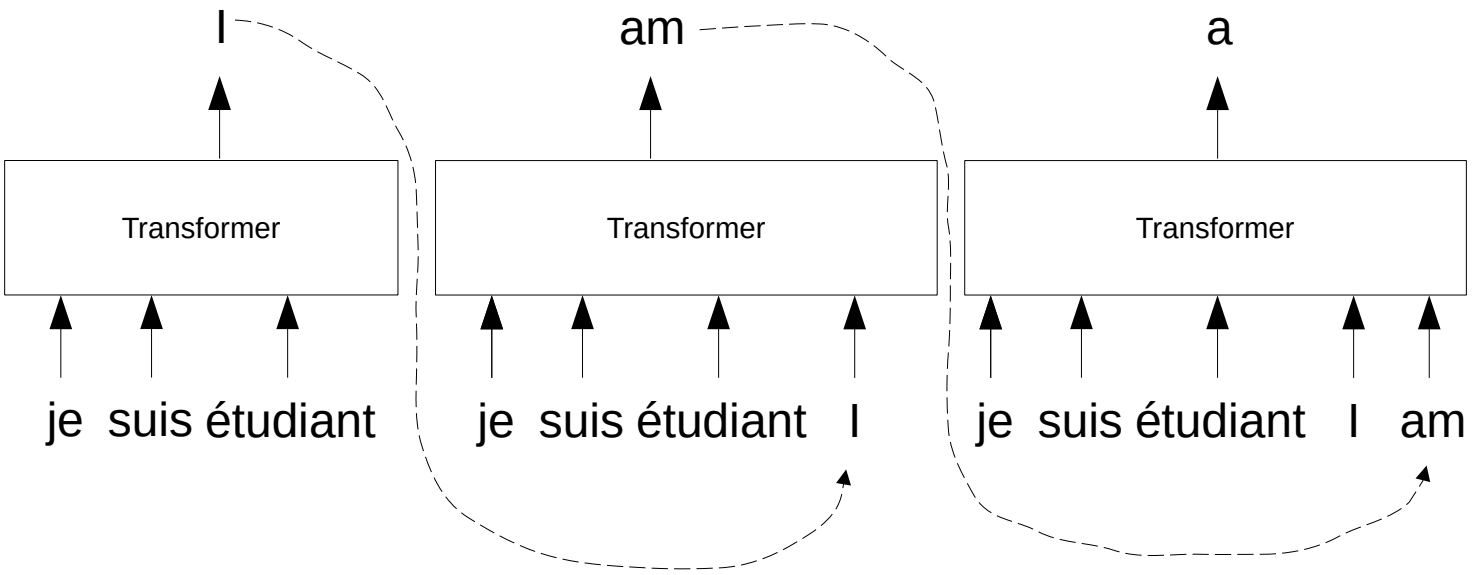
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



I)

Transformer (“decoder only”) - inférence

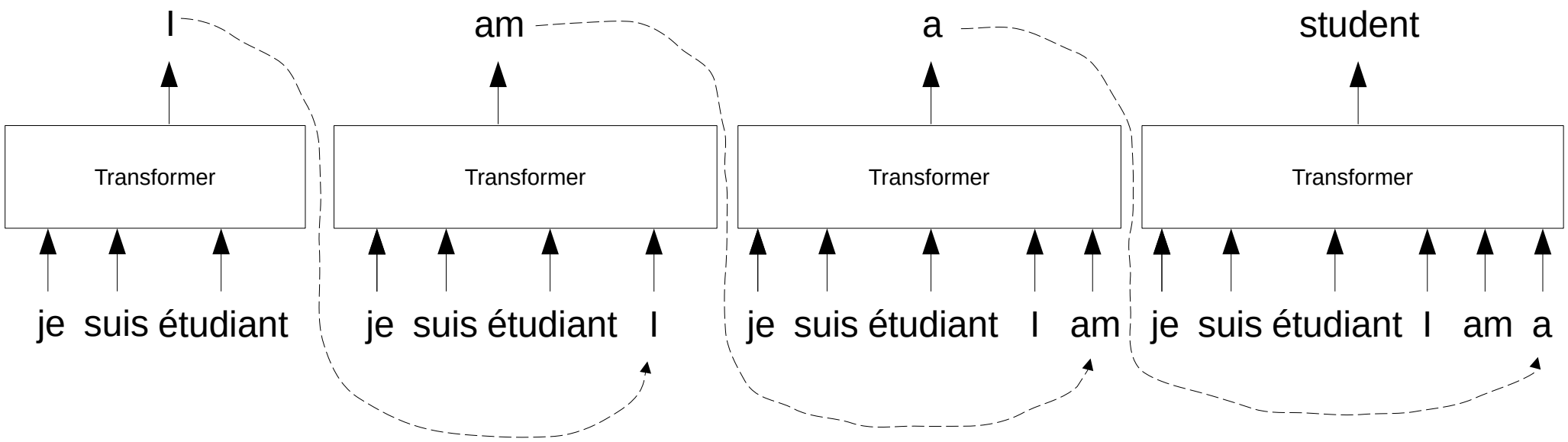
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



I)

Transformer (“decoder only”) - inférence

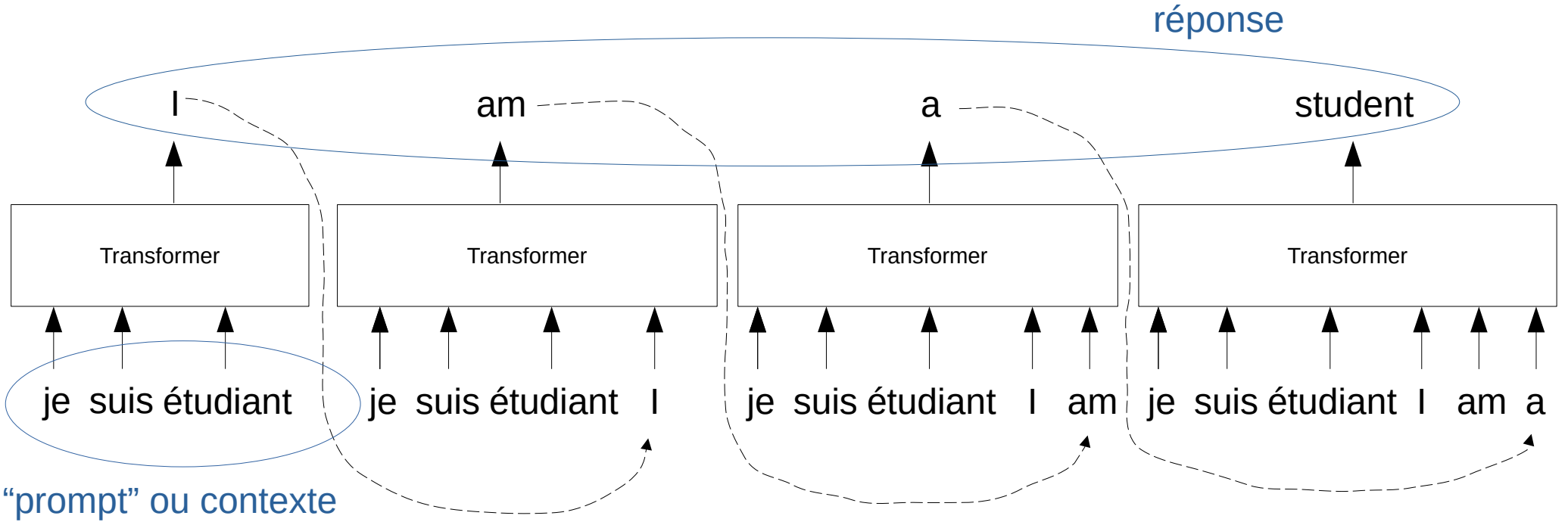
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



I)

Transformer (“decoder only”) - inférence

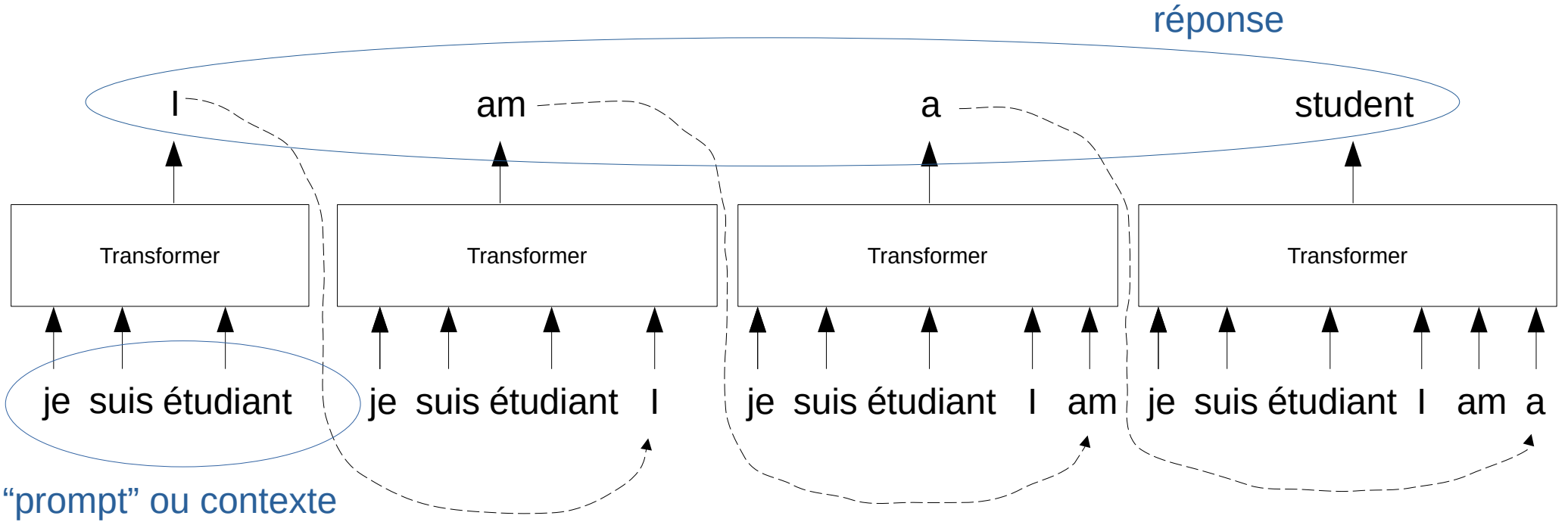
Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



I)

Transformer (“decoder only”) - inférence

Exemple : Traduction de “je suis étudiant” en anglais en “next token prediction”



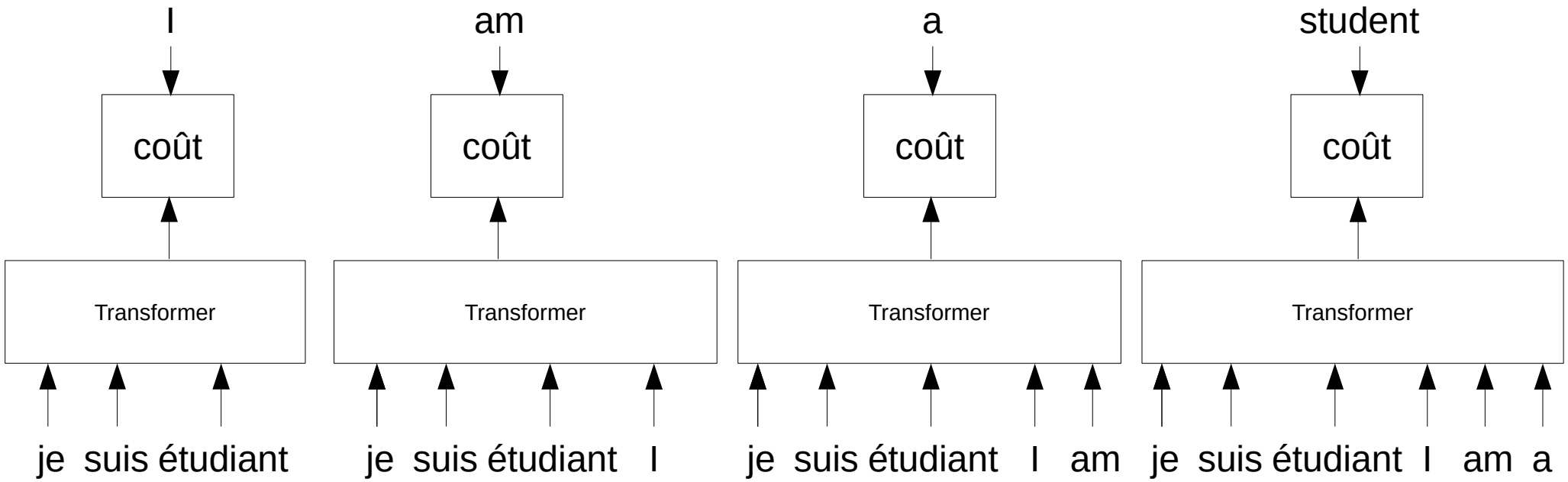
Remarque : Pas d’état caché à propager !

Inférence séquentielle (à cause du paradigme “next token prediction”)
→ Le Transformer ne règle pas ce problème (ChatGPT est séquentiel)

I)

Transformer (“decoder only”) - apprentissage

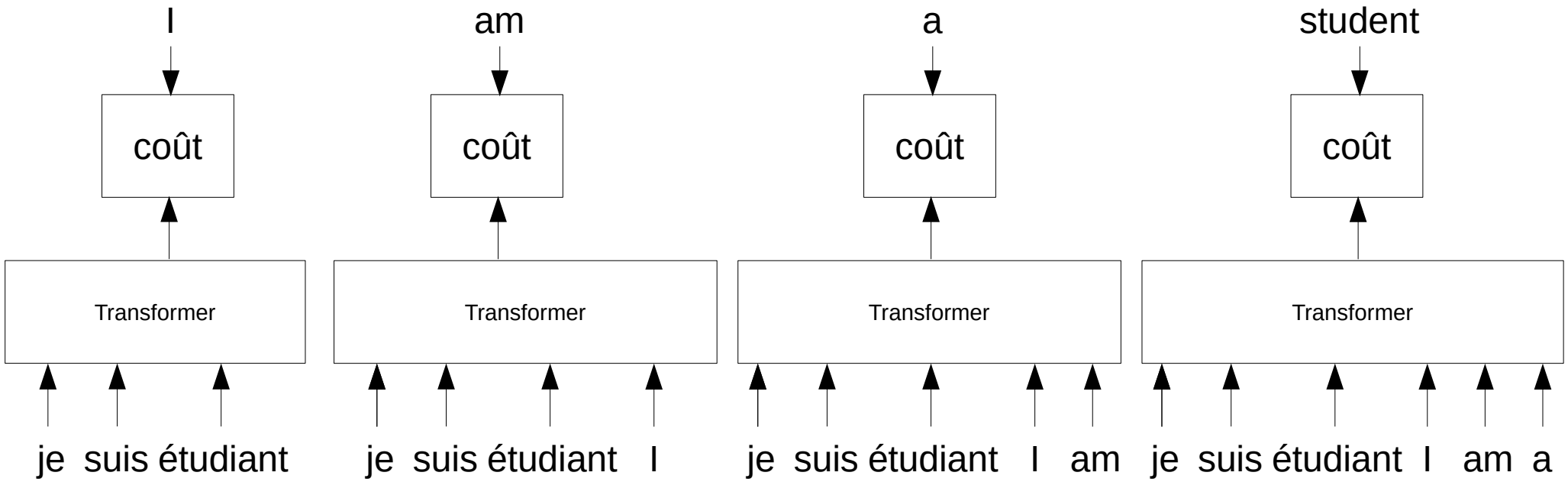
“Teacher forcing”



I)

Transformer (“decoder only”) - apprentissage

“Teacher forcing”

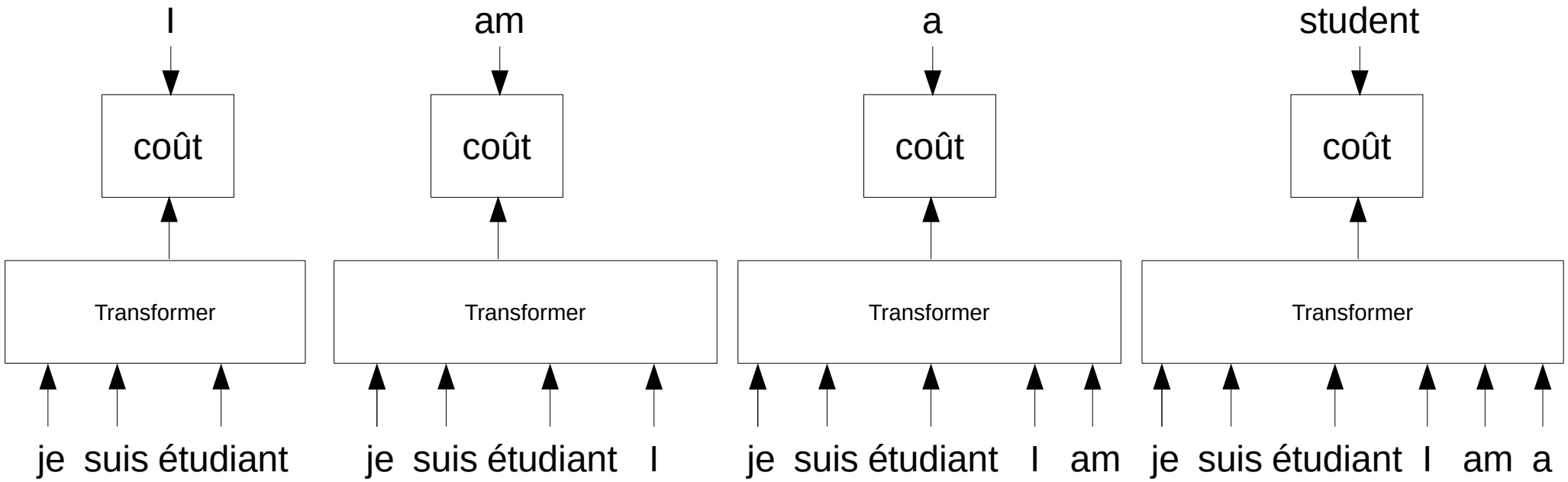


- Parallélisable à l’apprentissage car il n’y a plus d’état caché d’un instant à l’autre.

I)

Transformer (“decoder only”) - apprentissage

“Teacher forcing”



- Parallélisable à l’apprentissage car il n’y a plus d’état caché d’un instant à l’autre.

- Plus de problèmes de longue dépendance car toutes les entrées passées sont placées en entrée à chaque instant (il n’y a plus d’état caché !)

II) Couche d’attention à softmax

II)

Attention utilisant la fonction softmax

Entrées

$\mathbf{x}$: vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1...N_y}$: ensemble de vecteurs de dimension $1 \times D$

II)

Attention utilisant la fonction softmax

Entrées

$\mathbf{x}$: vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1\dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction

$$\frac{\exp(\mathbf{x}\mathbf{y}_j^\top)}{\quad}$$

Produit scalaire + exp :

$\gg 1$ si $\mathbf{x}$ est “attiré“ par $\mathbf{y}_j$

$= 1$ si $\mathbf{x}$ orthogonal à $\mathbf{y}_j$

≈ 0 si $\mathbf{x}$ est “repoussé” par $\mathbf{y}_j$

II)

Attention utilisant la fonction softmax

Entrées

$\mathbf{x}$: vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1...N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction

$$\frac{\exp(\mathbf{x}\mathbf{y}_j^\top)}{\quad}$$

Produit scalaire + exp :

$\gg 1$ si $\mathbf{x}$ est “attiré“ par $\mathbf{y}_j$

$= 1$ si $\mathbf{x}$ orthogonal à $\mathbf{y}_j$

≈ 0 si $\mathbf{x}$ est “repoussé” par $\mathbf{y}_j$

$\mathbf{y}_j$ “attire l’attention de” $\mathbf{x}$

II)

Attention utilisant la fonction softmax

Entrées

$\mathbf{x}$: vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1\dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction

$$\frac{\exp(\mathbf{x}\mathbf{y}_j^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x}\mathbf{y}_k^\top)}$$

Softmax

II)

Attention utilisant la fonction softmax

Entrées

$\mathbf{x}$: vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1\dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction

$$\mathbf{x}' = \sum_{j=1}^{N_y} \frac{\exp(\mathbf{x}\mathbf{y}_j^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x}\mathbf{y}_k^\top)} \mathbf{y}_j$$

Combinaison linéaire des $\{\mathbf{y}_i\}_{i=1\dots N_y}$

Les poids les plus élevés de cette combinaison linéaire correspondent aux vecteurs ayant le plus “attiré l’attention” de $\mathbf{x}$

II)

Attention utilisant la fonction softmax

Entrées

$\mathbf{x}$: vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1\dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction

$$\mathbf{x}' = \sum_{j=1}^{N_y} \frac{\exp\left(\mathbf{x}\mathtt{Q}(\mathbf{y}_j\mathtt{K})^\top\right)}{\sum_{k=1}^{N_y} \exp\left(\mathbf{x}\mathtt{Q}(\mathbf{y}_k\mathtt{K})^\top\right)} \mathbf{y}_j$$

Pour pouvoir apprendre à “attirer l’attention” ⟶ introduction de paramètres à optimiser

Paramètres

$\mathtt{Q}$: matrice “query” de taille $D \times L$

$\mathtt{K}$: matrice “key” de taille $D \times L$

II)

Attention utilisant la fonction softmax

Entrées

$\mathbf{x}$: vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1\dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction

$$\mathbf{x}' = \mathbf{x} + \sum_{j=1}^{N_y} \frac{\exp(\mathbf{x}\mathtt{Q}(\mathbf{y}_j\mathtt{K})^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x}\mathtt{Q}(\mathbf{y}_k\mathtt{K})^\top)} \mathbf{y}_j\mathtt{V}$$

En pratique, la combinaison linéaire est elle-même transformée linéairement, et suivie d'une connection résiduelle.

Paramètres

$\mathtt{Q}$: matrice "query" de taille $D \times L$

$\mathtt{K}$: matrice "key" de taille $D \times L$

$\mathtt{V}$: matrice "value" de taille $D \times D$

II)

Attention utilisant la fonction softmax

Entrées

$\mathbf{x}$: vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1\dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction

$$\mathbf{x}' = \mathbf{x} + \sum_{j=1}^{N_y} \frac{\exp(\mathbf{x}\mathtt{Q}(\mathbf{y}_j\mathtt{K})^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x}\mathtt{Q}(\mathbf{y}_k\mathtt{K})^\top)} \mathbf{y}_j\mathtt{V}$$

Paramètres

$\mathtt{Q}$: matrice “query” de taille $D \times L$

$\mathtt{K}$: matrice “key” de taille $D \times L$

$\mathtt{V}$: matrice “value” de taille $D \times D$

Terminologie : “Dot-product attention”, plus rarement “softmax attention”

II)

Couche d’inter-attention softmax (“Cross-attention”)

Entrées

$\{\mathbf{x}_i\}_{i=1\dots N_x}$: vecteurs de dimension D ⟶ $\mathtt{X}$: matrice de taille $N_x \times D$

$\{\mathbf{y}_i\}_{i=1\dots N_y}$: vecteurs de dimension D ⟶ $\mathtt{Y}$: matrice de taille $N_y \times D$

II)

Couche d’inter-attention softmax (“Cross-attention”)

Entrées

$\{\mathbf{x}_i\}_{i=1\dots N_x}$: vecteurs de dimension D ⟶ $\mathtt{X}$: matrice de taille $N_x \times D$

$\{\mathbf{y}_i\}_{i=1\dots N_y}$: vecteurs de dimension D ⟶ $\mathtt{Y}$: matrice de taille $N_y \times D$

Sorties $\{\mathbf{x}'_i\}_{i=1\dots N_x}$: vecteurs de dimension D ⟶ $\mathtt{X}'$: matrice de taille $N_x \times D$

II)

Couche d'inter-attention softmax ("Cross-attention")

Entrées

$\{\mathbf{x}_i\}_{i=1\dots N_x}$: vecteurs de dimension D ⟶ $\mathtt{X}$: matrice de taille $N_x \times D$

$\{\mathbf{y}_i\}_{i=1\dots N_y}$: vecteurs de dimension D ⟶ $\mathtt{Y}$: matrice de taille $N_y \times D$

Sorties $\{\mathbf{x}'_i\}_{i=1\dots N_x}$: vecteurs de dimension D ⟶ $\mathtt{X}'$: matrice de taille $N_x \times D$

$$\mathtt{M} = \mathtt{XQ}(\mathtt{YK})^\top$$

$$\mathtt{S} = \mathrm{softmax}(\mathtt{M}, \mathrm{dim}{=}1)$$

$$\mathtt{X}' = \mathtt{X} + \mathtt{SYV}$$

Calcul réalisé en parallèle sur les vecteurs $\{\mathbf{x}_i\}_{i=1\dots N_x}$

II)

Couche d'inter-attention softmax ("Cross-attention")

Entrées

$\{\mathbf{x}_i\}_{i=1\dots N_x}$: vecteurs de dimension D ⟶ $\mathtt{X}$: matrice de taille $N_x \times D$

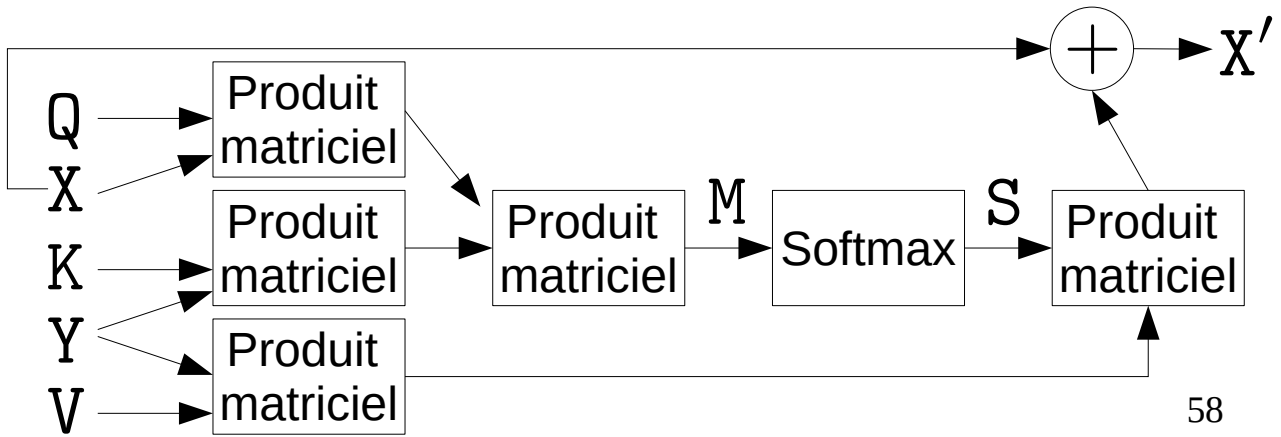
$\{\mathbf{y}_i\}_{i=1\dots N_y}$: vecteurs de dimension D ⟶ $\mathtt{Y}$: matrice de taille $N_y \times D$

Sorties $\{\mathbf{x}'_i\}_{i=1\dots N_x}$: vecteurs de dimension D ⟶ $\mathtt{X}'$: matrice de taille $N_x \times D$

$$\mathtt{M} = \mathtt{XQ}(\mathtt{YK})^\top$$

$$\mathtt{S} = \mathrm{softmax}(\mathtt{M}, \mathrm{dim}{=}1)$$

$$\mathtt{X}' = \mathtt{X} + \mathtt{SYV}$$



Calcul réalisé en parallèle sur les vecteurs $\{\mathbf{x}_i\}_{i=1\dots N_x}$

II)

Couche d'inter-attention softmax ("Cross-attention") (suite)

$$\mathtt{X} : N_x \times D \quad \mathtt{Y} : N_y \times D \quad \mathtt{X}' : N_x \times D$$

$$\mathtt{M} = \mathtt{XQ}(\mathtt{YK})^\top : N_x \times N_y \qquad \mathtt{S} = \mathrm{softmax}(\mathtt{M}, \mathrm{dim}{=}1) : N_x \times N_y$$

II)

Couche d'inter-attention softmax ("Cross-attention") (suite)

$$\mathtt{X} : N_x \times D \quad \mathtt{Y} : N_y \times D \quad \mathtt{X}' : N_x \times D$$

$$\mathtt{M} = \mathtt{XQ}(\mathtt{YK})^\top : N_x \times N_y \qquad \mathtt{S} = \mathrm{softmax}(\mathtt{M}, \mathrm{dim}{=}1) : N_x \times N_y$$

Calcul

- Nombre d'opérations potentiellement très élevé

+ Parallélisable

Stockage

- Mémoire requise pour stocker M et S potentiellement très élevée

II)

Couche d'inter-attention softmax ("Cross-attention") (suite)

$$\mathtt{X} : N_x \times D \quad \mathtt{Y} : N_y \times D \quad \mathtt{X}' : N_x \times D$$

$$\mathtt{M} = \mathtt{XQ}(\mathtt{YK})^\top : N_x \times N_y \qquad \mathtt{S} = \mathrm{softmax}(\mathtt{M}, \mathrm{dim}{=}1) : N_x \times N_y$$

Calcul

- Nombre d'opérations potentiellement très élevé

+ Parallélisable

Stockage

- Mémoire requise pour stocker M et S potentiellement très élevée

Exemple : $N_x = N_y = 640 \times 480 \approx 3.10^5$ pixels

$N_x \times N_y \approx 9.10^{10}$ flottants (32 bits) $\longrightarrow 360\mathrm{Go}$

II)

Cas particulier : Couche d'auto-attention ("Self-attention")

Entrées $\{\mathbf{x}_i\}_{i=1\dots N_x}$: vecteurs de dimension D ⟶ $\mathtt{X}$: matrice de taille $N_x \times D$

Sorties $\{\mathbf{x}'_i\}_{i=1\dots N_x}$: vecteurs de dimension D ⟶ $\mathtt{X}'$: matrice de taille $N_x \times D$

II)

Cas particulier : Couche d'auto-attention ("Self-attention")

Entrées $\{\mathbf{x}_i\}_{i=1\dots N_x}$: vecteurs de dimension D ⟶ $\mathtt{X}$: matrice de taille $N_x \times D$

Sorties $\{\mathbf{x}'_i\}_{i=1\dots N_x}$: vecteurs de dimension D ⟶ $\mathtt{X}'$: matrice de taille $N_x \times D$

$$\mathtt{M} = \mathtt{XQ}(\mathtt{XK})^\top \ : \ N_x \times N_x$$
$$\mathtt{S} = \mathrm{softmax}(\mathtt{M}, \mathrm{dim}{=}1) \ : \ N_x \times N_x$$
$$\mathtt{X}' = \mathtt{X} + \mathtt{SXV} \ : \ N_x \times D$$

Transfert d'information depuis $\mathtt{X}$ vers lui-même, le tout stocké dans $\mathtt{X}'$

II)

Attention softmax à têtes multiples
”Multi-head dot-product attention”

Entrées

$\mathbf{x}$: vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1\ldots N_y}$: ensemble de vecteurs de dimension $1 \times D$

---

1 tête = 1 façon
“d’attirer l’attention”

$$\mathbf{x}' = \mathbf{x} + \sum_{j=1}^{N_y} \frac{\exp\left(\mathbf{x}\mathtt{Q}(\mathbf{y}_j\mathtt{K})^\top\right)}{\sum_{k=1}^{N_y} \exp\left(\mathbf{x}\mathtt{Q}(\mathbf{y}_k\mathtt{K})^\top\right)} \mathbf{y}_j\mathtt{V}$$

II)

Attention softmax à têtes multiples
”Multi-head dot-product attention”

Entrées

$\mathbf{x}$: vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1\ldots N_y}$: ensemble de vecteurs de dimension $1 \times D$

---

1 tête = 1 façon
“d’attirer l’attention”

$$\mathbf{x}' = \mathbf{x} + \sum_{j=1}^{N_y} \frac{\exp\left(\mathbf{x}\mathtt{Q}(\mathbf{y}_j\mathtt{K})^\top\right)}{\sum_{k=1}^{N_y} \exp\left(\mathbf{x}\mathtt{Q}(\mathbf{y}_k\mathtt{K})^\top\right)} \mathbf{y}_j\mathtt{V}$$

H têtes = H façons
“d’attirer l’attention”

$$\mathbf{x}' = \mathbf{x} + \sum_{h=1}^{H}\sum_{j=1}^{N_y} \frac{\exp\left(\mathbf{x}\mathtt{Q}_h(\mathbf{y}_j\mathtt{K}_h)^\top\right)}{\sum_{k=1}^{N_y} \exp\left(\mathbf{x}\mathtt{Q}_h(\mathbf{y}_k\mathtt{K}_h)^\top\right)} \mathbf{y}_j\mathtt{V}_h$$

II)

Attention softmax à têtes multiples (suite)
”Multi-head dot-product attention”

Entrées

$\mathbf{x}$: vecteur de dimension $1 \times D$

$\{\mathbf{y}_i\}_{i=1\dots N_y}$: ensemble de vecteurs de dimension $1 \times D$

Fonction

$$\mathbf{x}' = \mathbf{x} + \sum_{h=1}^{H} \sum_{j=1}^{N_y} \frac{\exp(\mathbf{x}\mathtt{Q}_h(\mathbf{y}_j\mathtt{K}_h)^\top)}{\sum_{k=1}^{N_y} \exp(\mathbf{x}\mathtt{Q}_h(\mathbf{y}_k\mathtt{K}_h)^\top)} \mathbf{y}_j\mathtt{V}_h\mathtt{W}_h$$

Paramètres

$\{\mathtt{Q}_h\}_{h=1\dots H}$: matrices “query” de taille $D \times L$

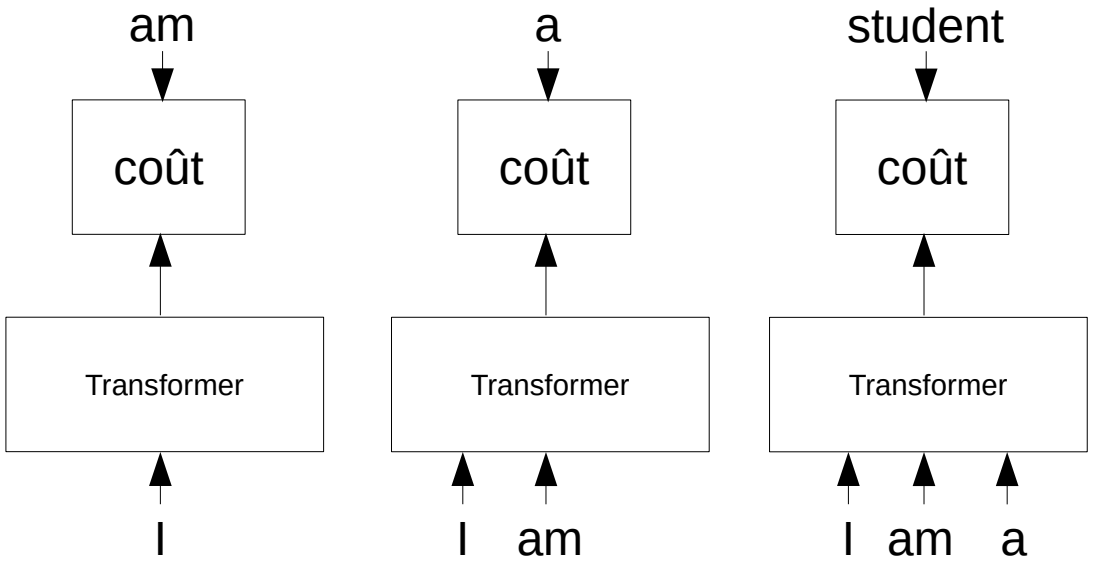
$\{\mathtt{K}_h\}_{h=1\dots H}$: matrices “key” de taille $D \times L$

$\{\mathtt{V}_h\}_{h=1\dots H}$: matrices “value” de taille $D \times L$

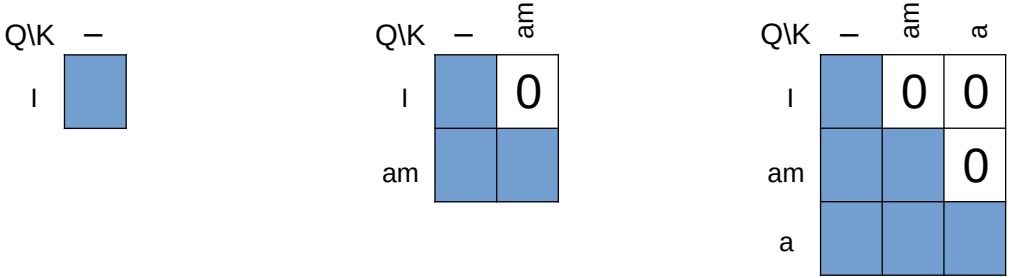
$\{\mathtt{W}_h\}_{h=1\dots H}$: matrices “output” de taille $L \times D$

II)

“Masked self-attention”

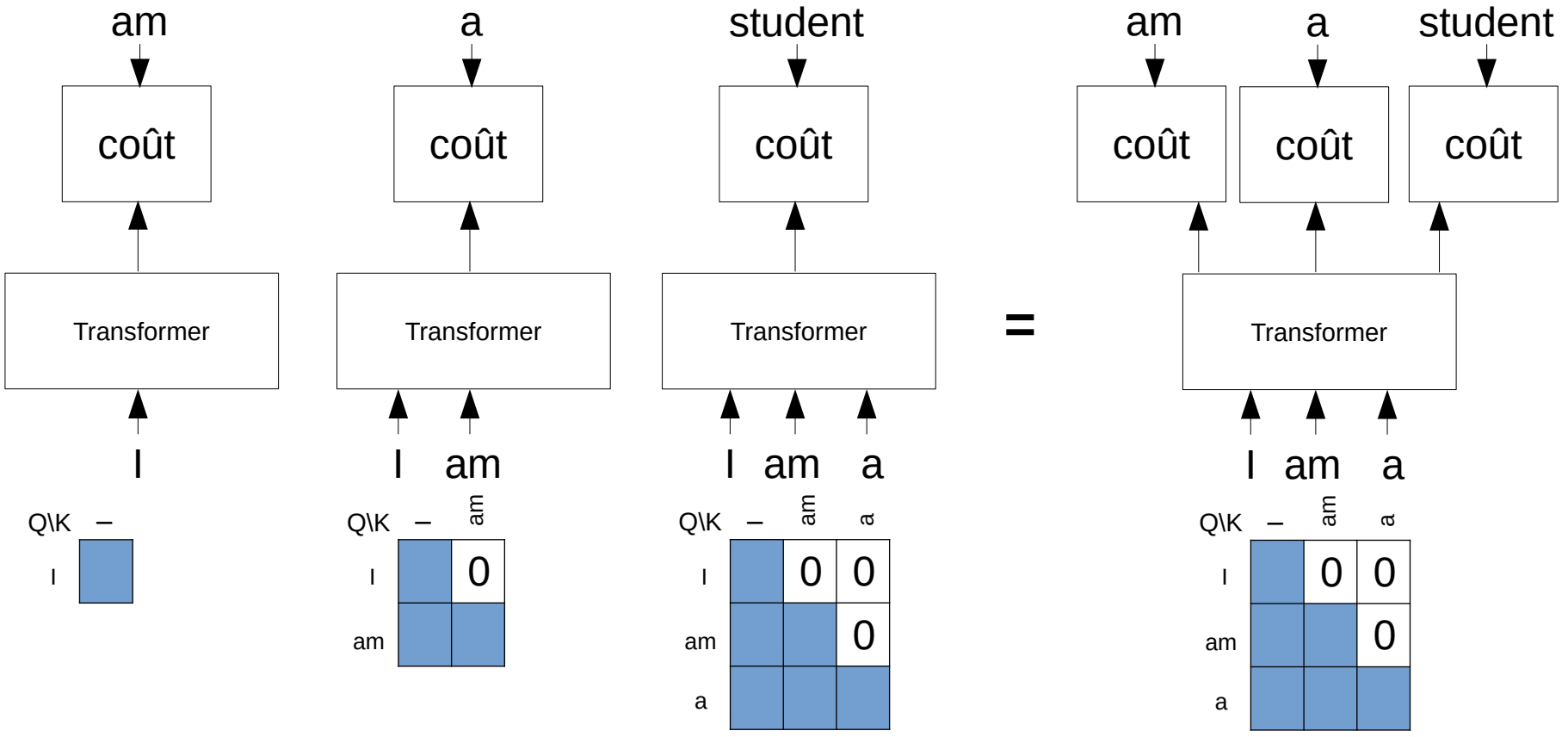


Matrices d’attention Post-softmax (dans chaque couche de self-attention)



II)

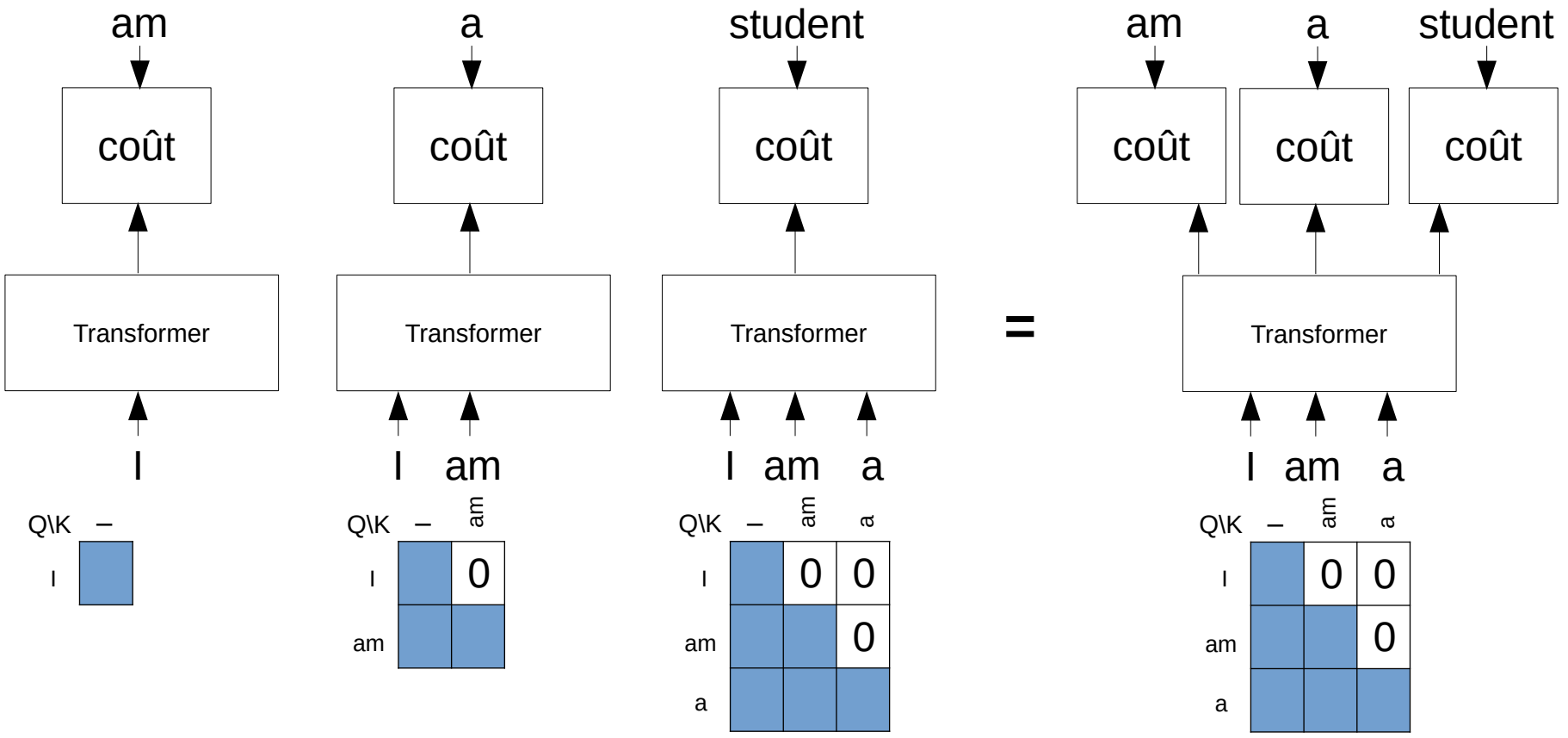
“Masked self-attention”



Matrices d’attention Post-softmax (dans chaque couche de self-attention)

II)

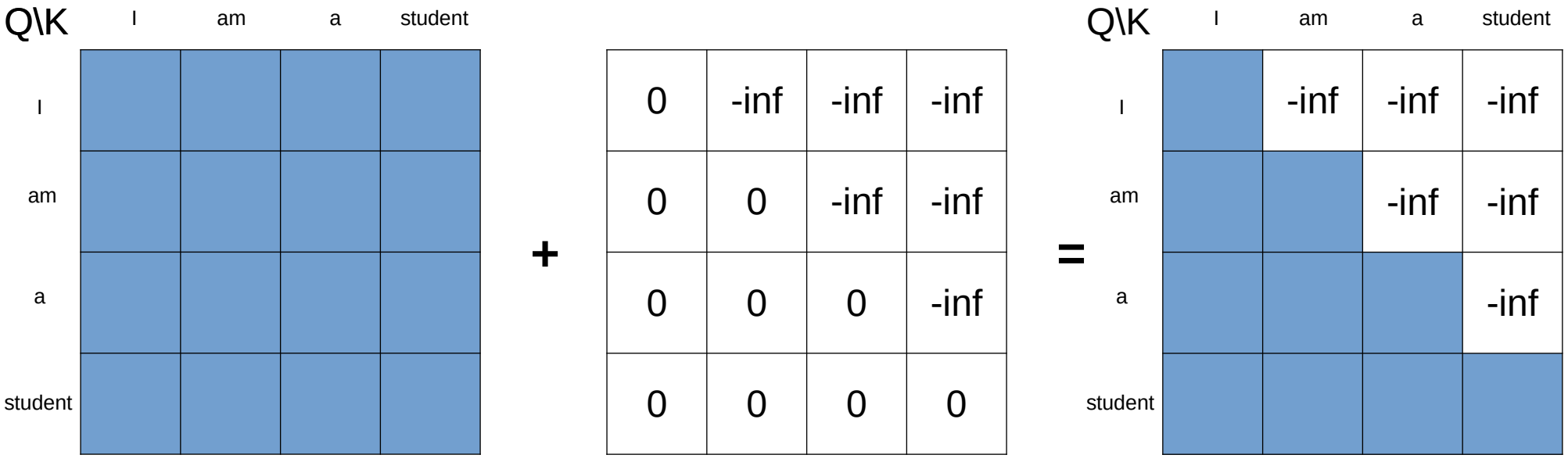
“Masked self-attention”



Masquer le “futur” permet un apprentissage très efficace (et l’utilisation du “KV cache” à l’inférence).

II)

“Masked self-attention” (suite)



Matrice d’attention pré-softmax

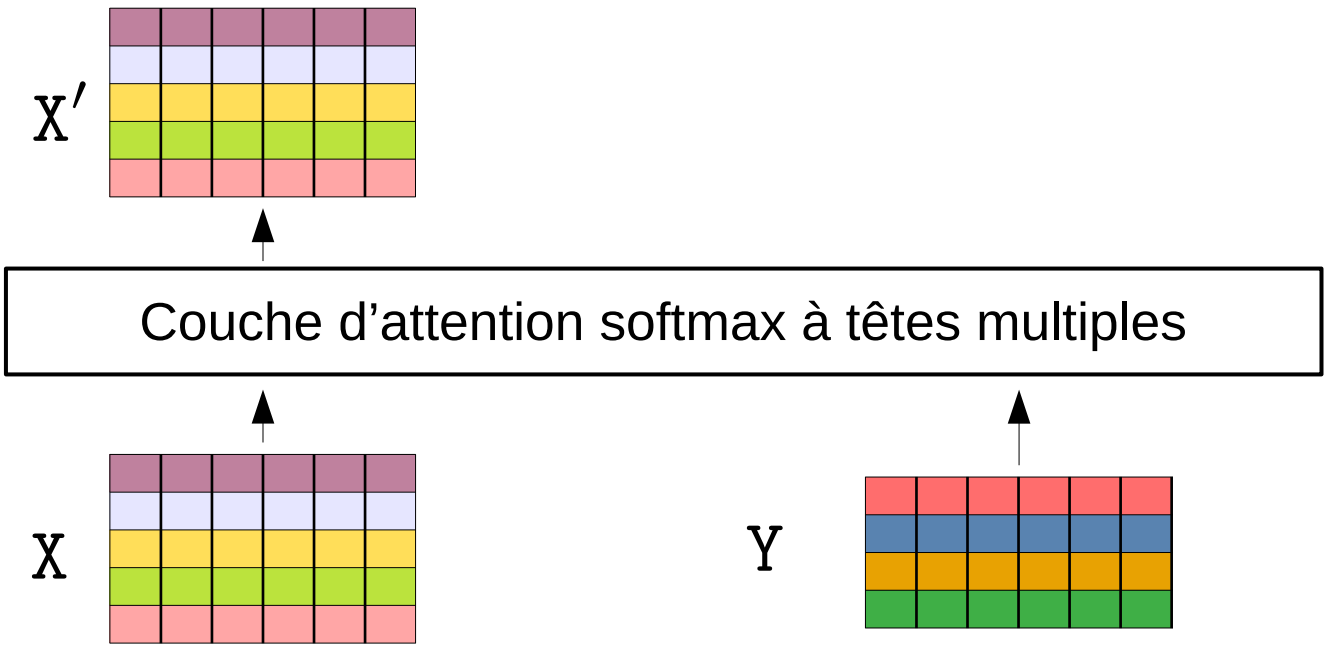
Masque causal

Matrice d’attention causale pré-softmax

II)

Bloc d’attention “classique”

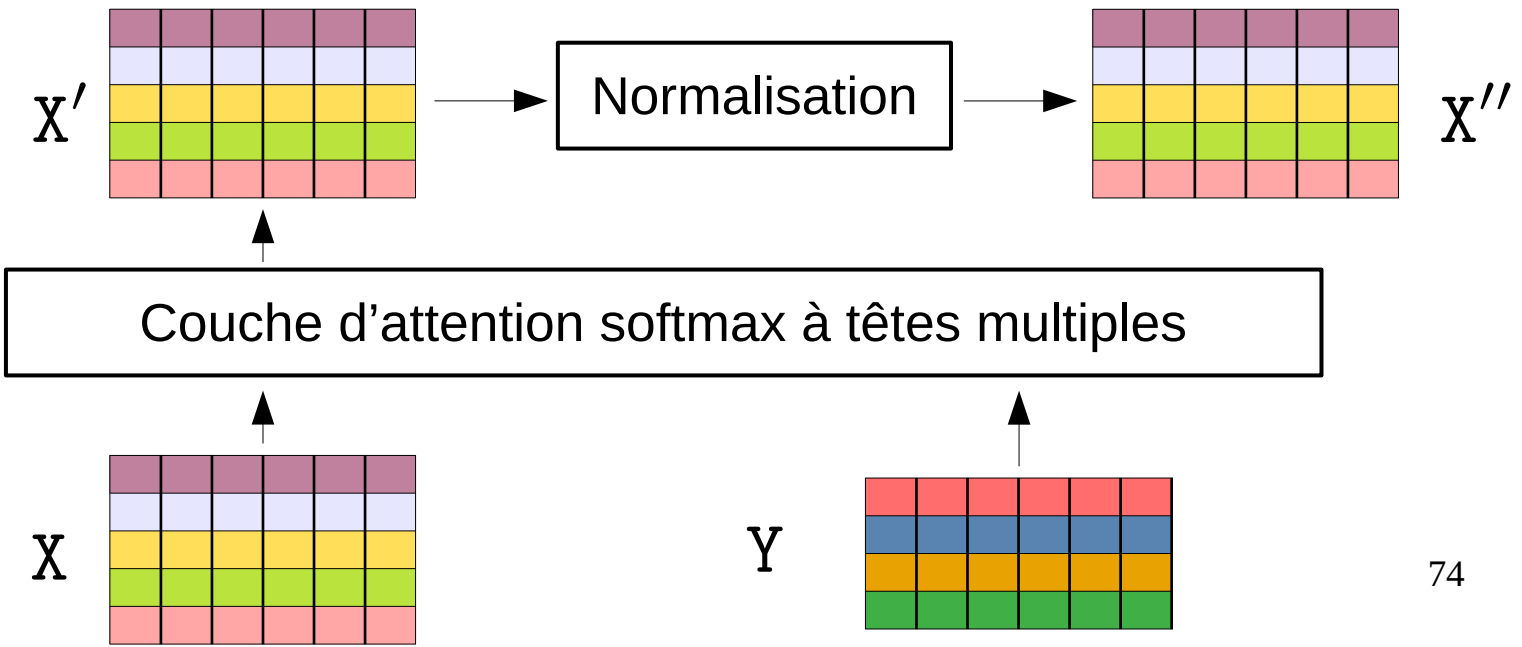
Transfert d’information de Y vers X



II)

Bloc d'attention "classique"

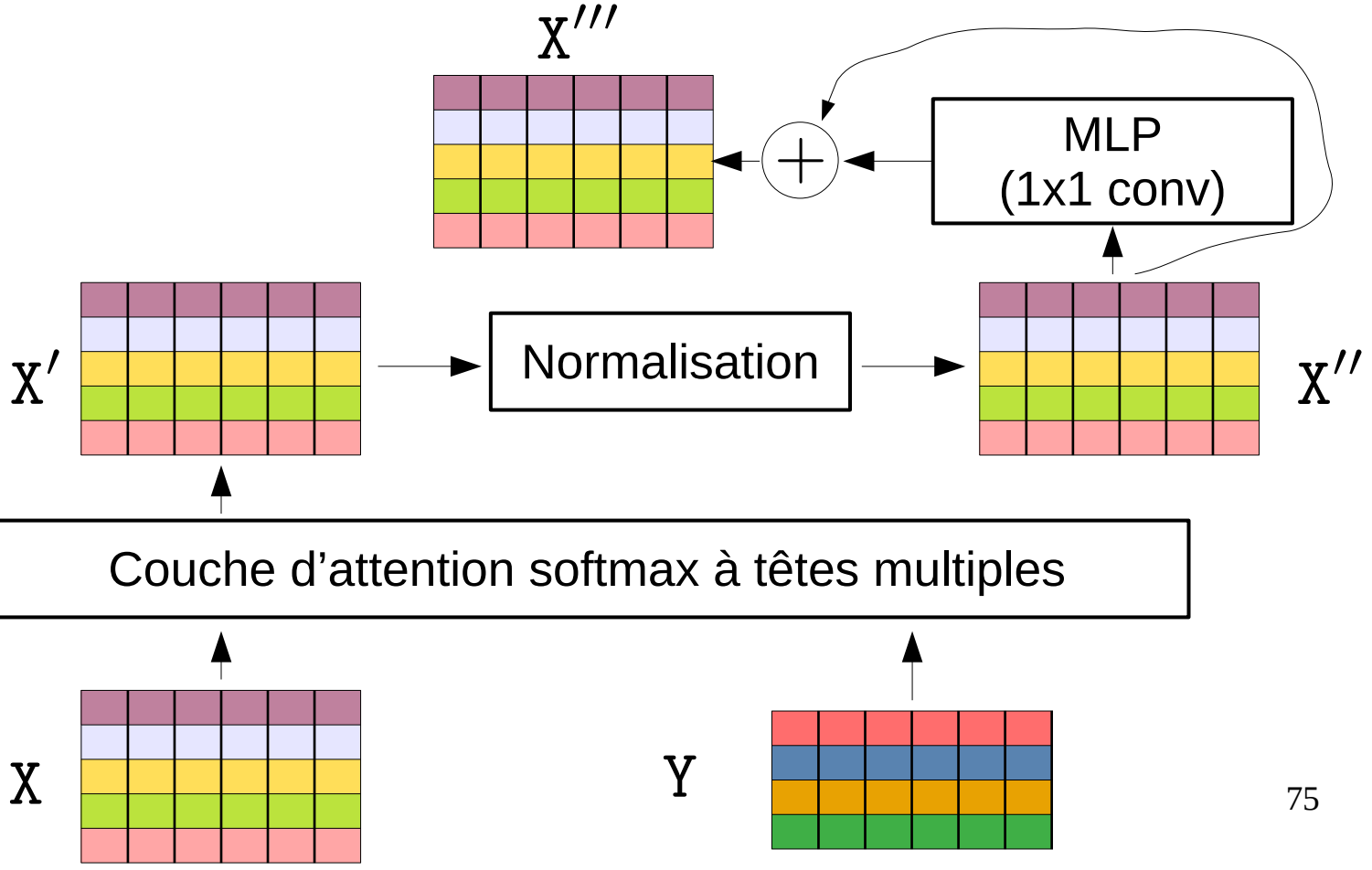
Transfert d'information de Y vers X



II)

Bloc d'attention "classique"

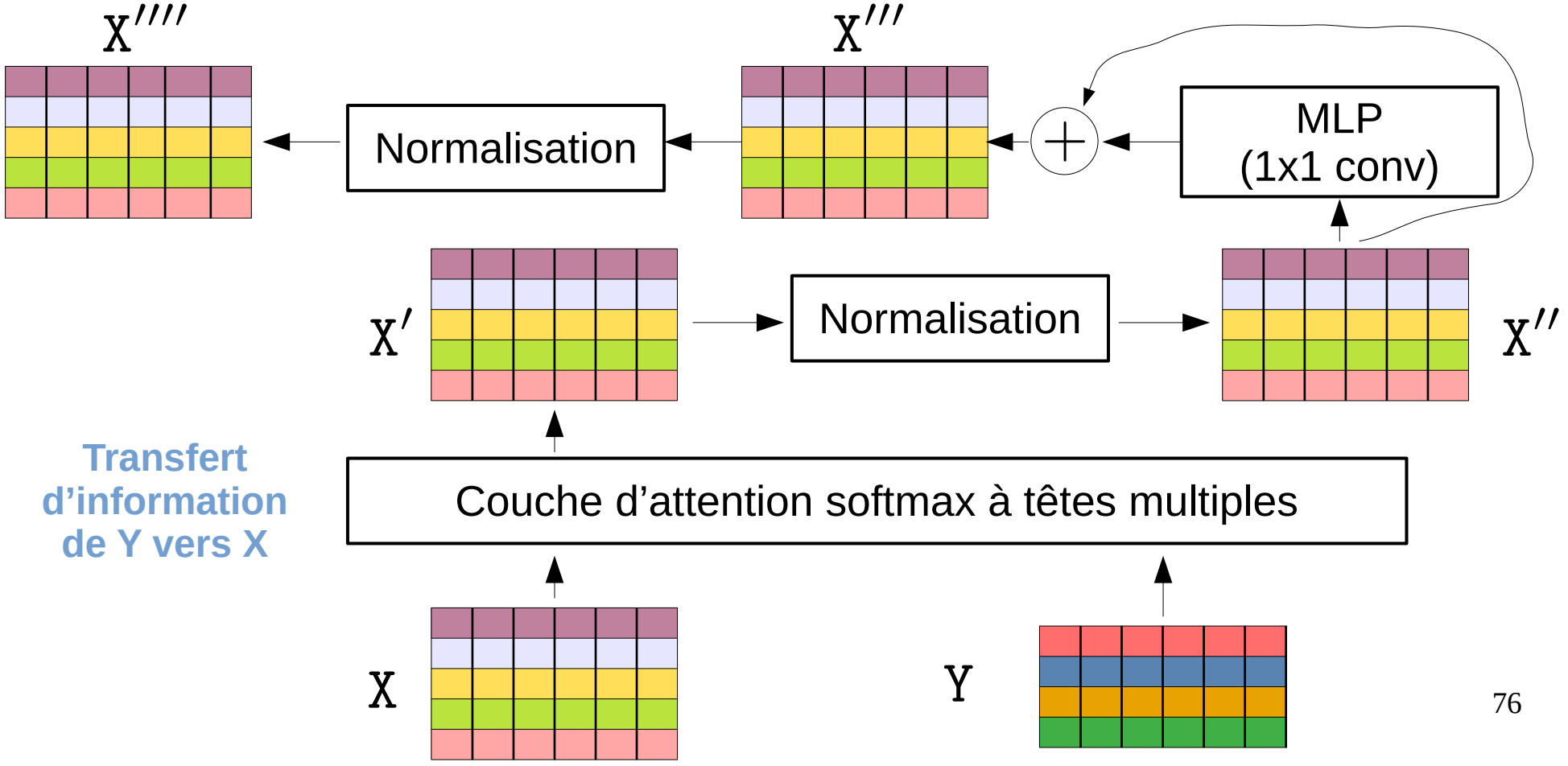
Transformation non-linéaire de chaque ligne



Transfert d'information de Y vers X

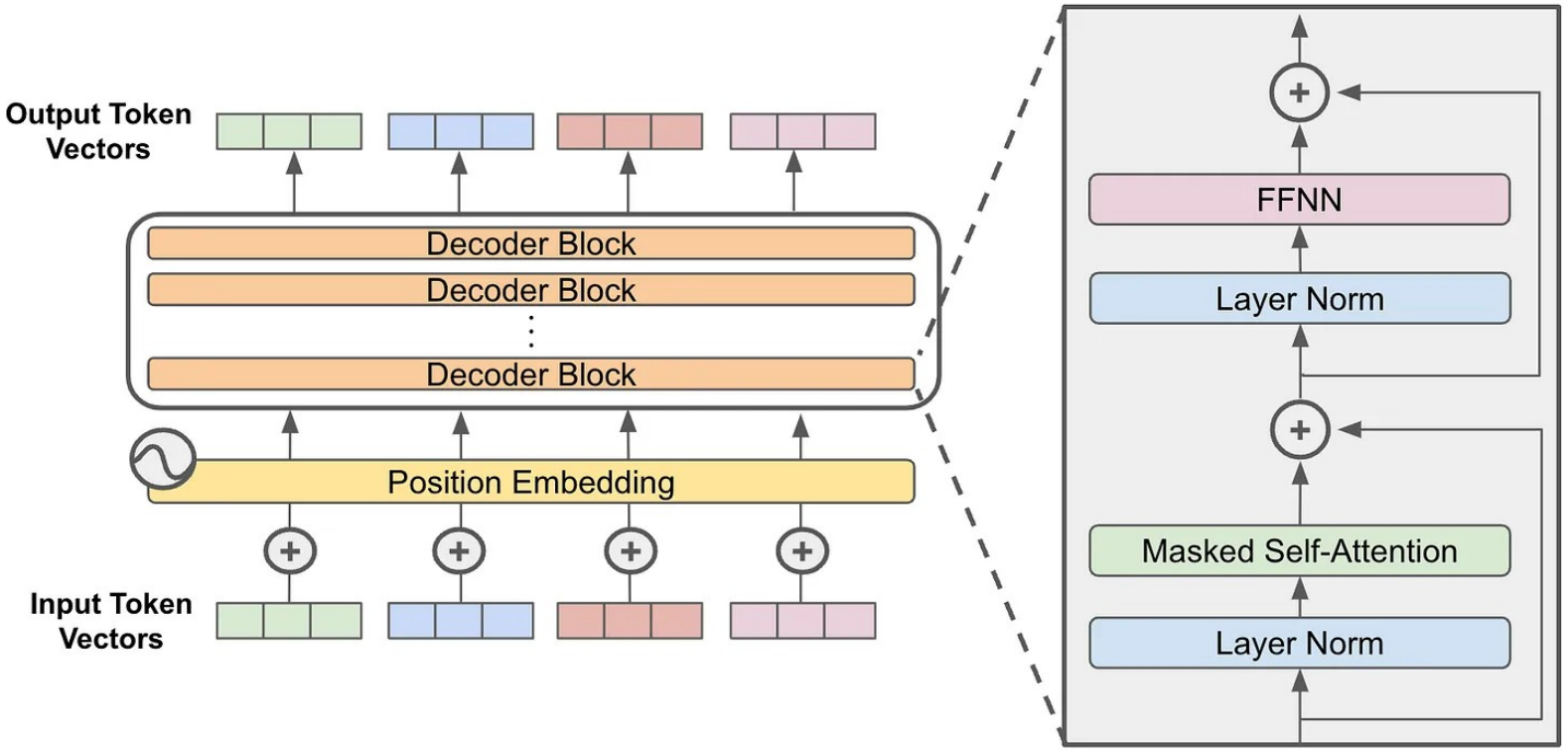
II)

Bloc d'attention "classique"



II)

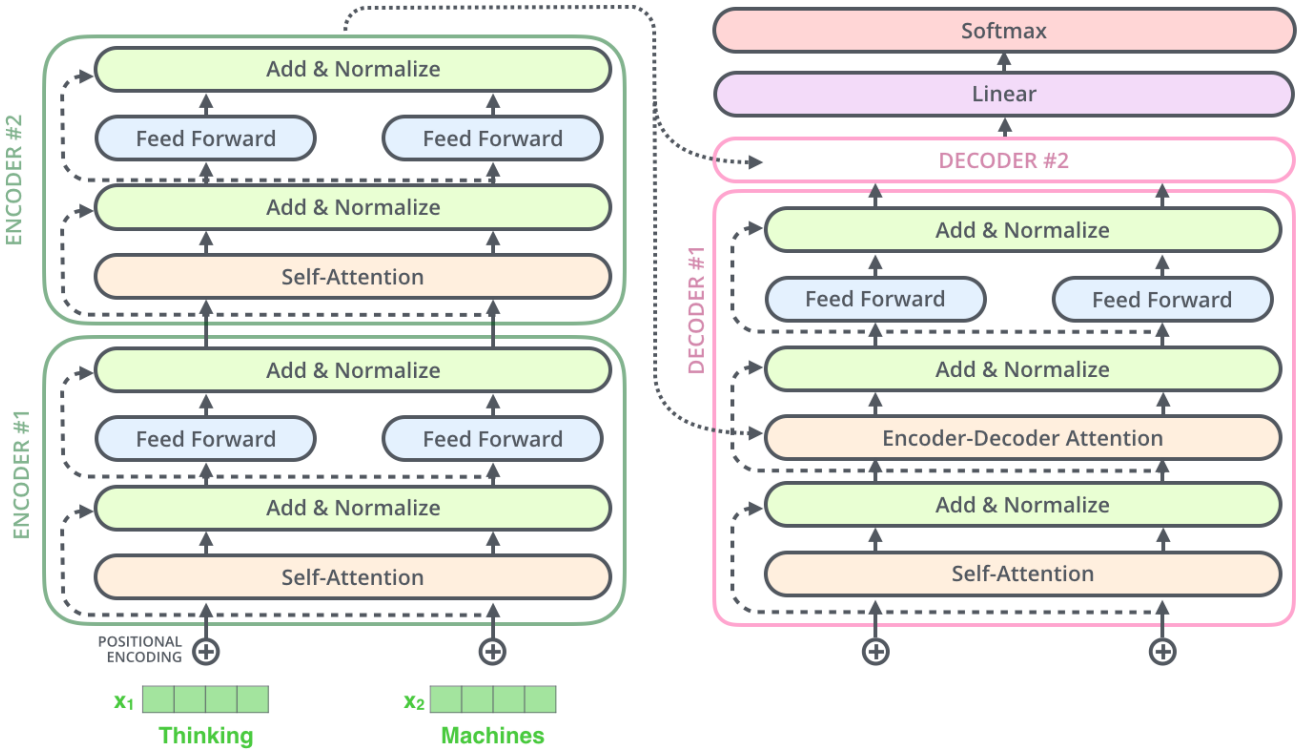
Vue détaillée d’un Transformer “Decoder-Only”



Remarque : pas de “cross-attention”, seulement des (masked) “self-attention”

II)

Vue détaillée d’un Transformer “Encoder-Decoder”

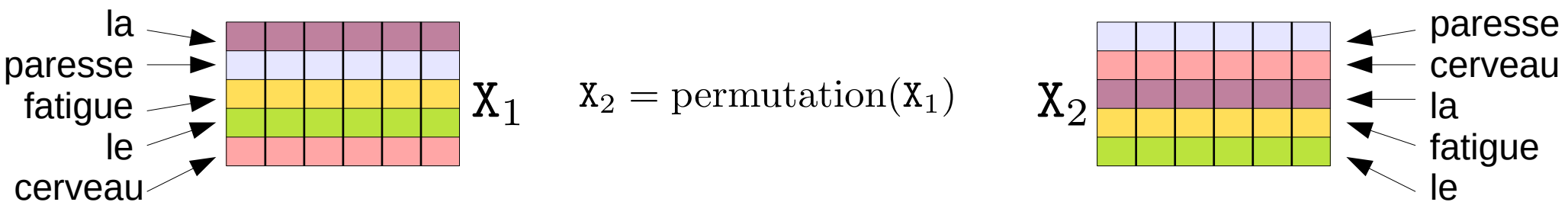


Remarque : “Encoder-Decoder Attention” = “cross-attention”

III) Équivariance par permutation et encodage de la position

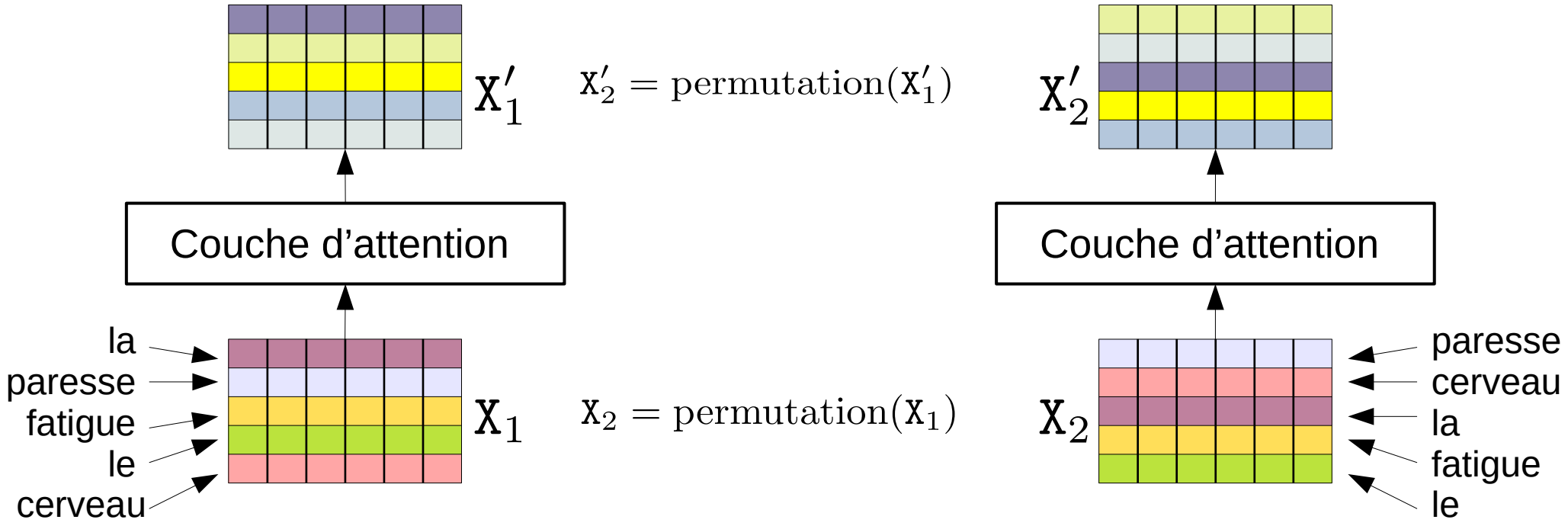
III)

Équivariance par permutation



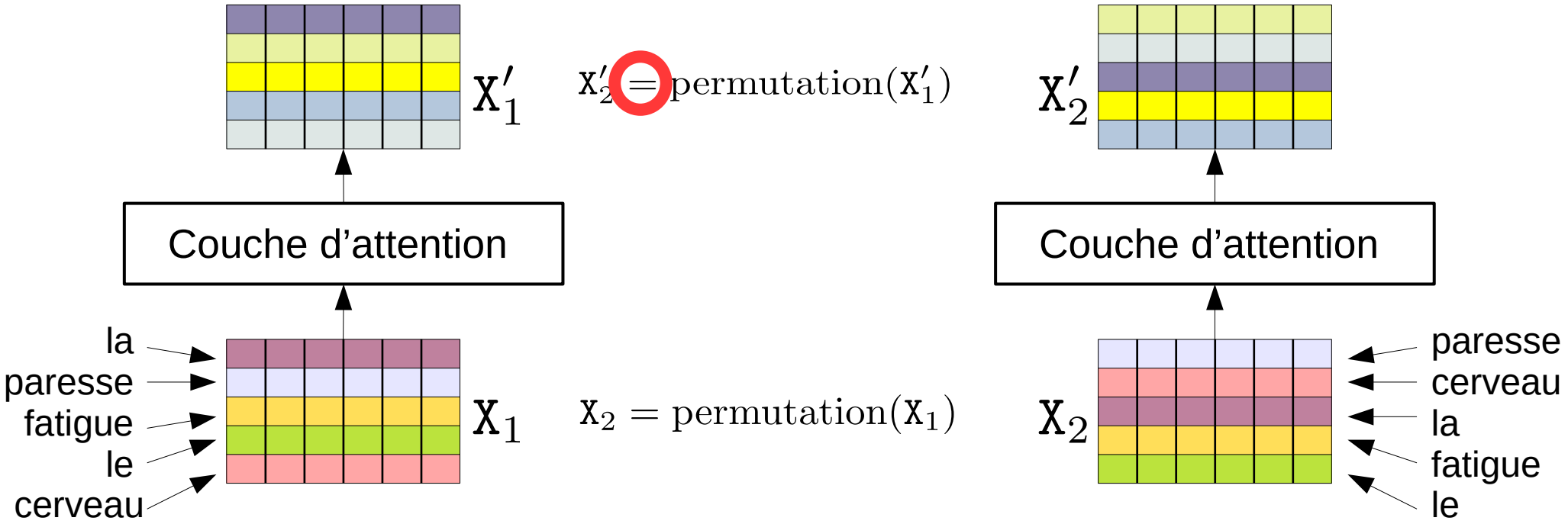
III)

Équivariance par permutation



III)

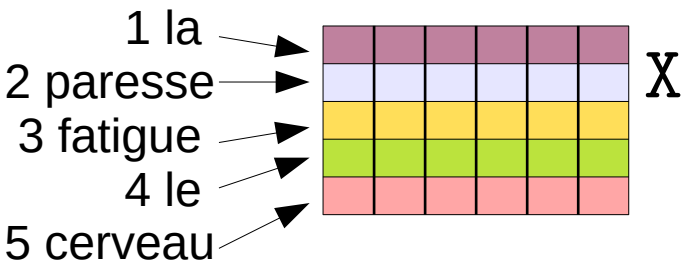
Équivariance par permutation



Pour un ensemble ordonné (phrase, signal, image, etc.)
→ necessité d'encoder la position

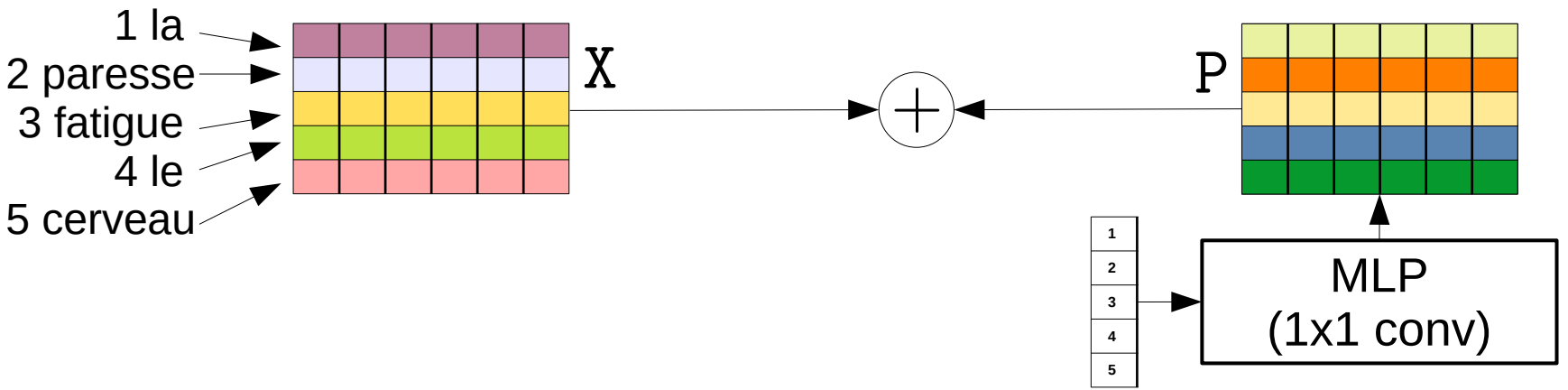
III)

Encodage de la position



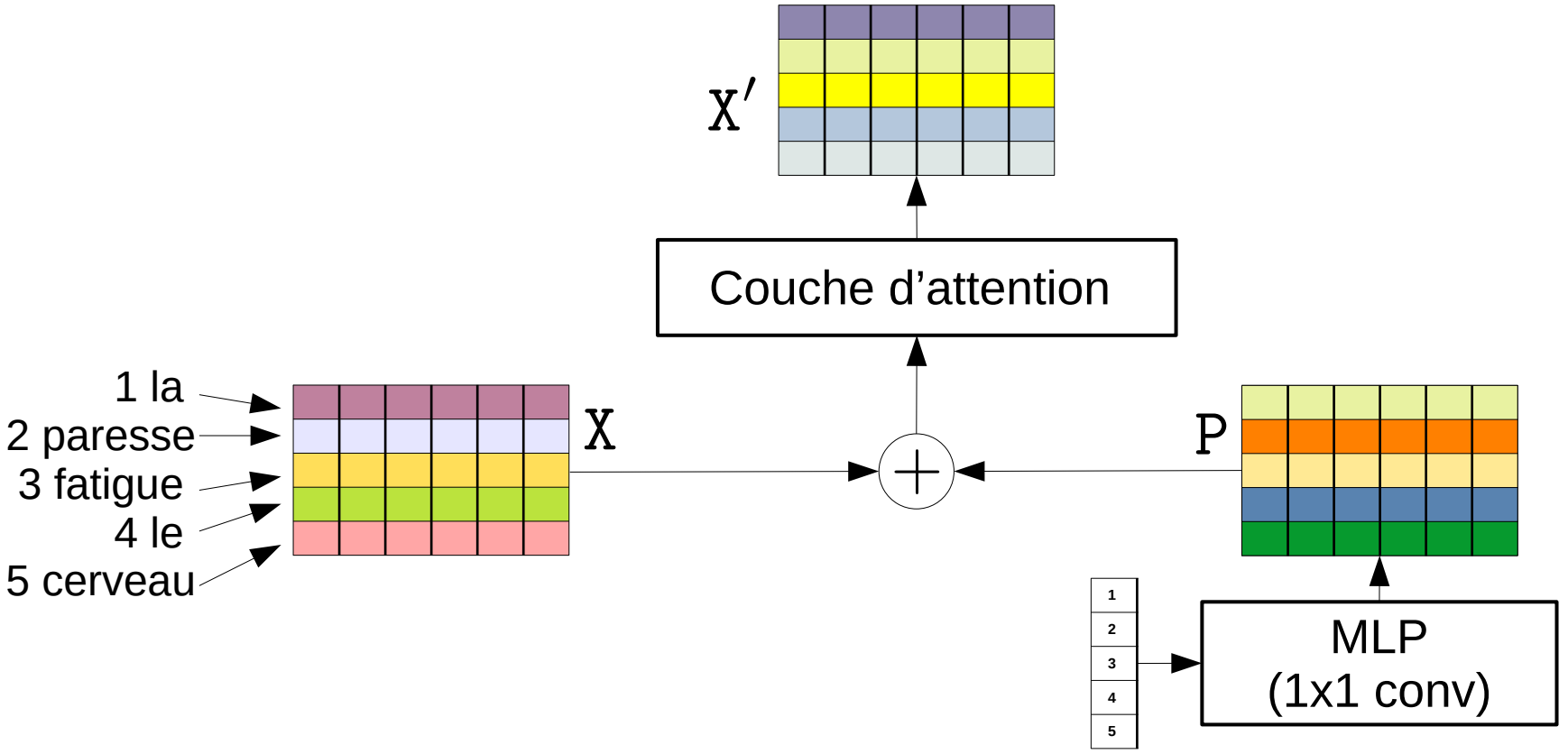
III)

Encodage de la position



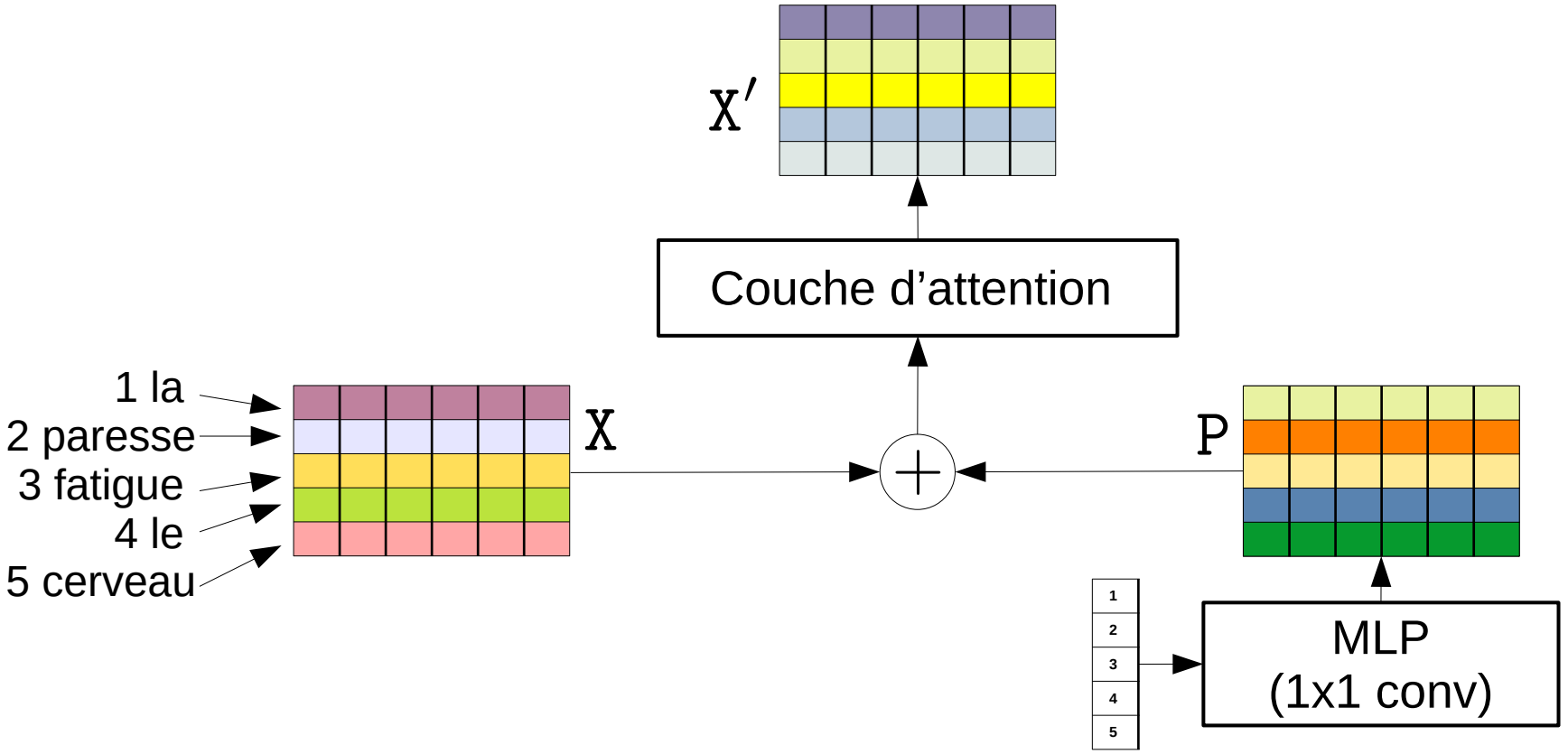
III)

Encodage de la position



III)

Encodage de la position

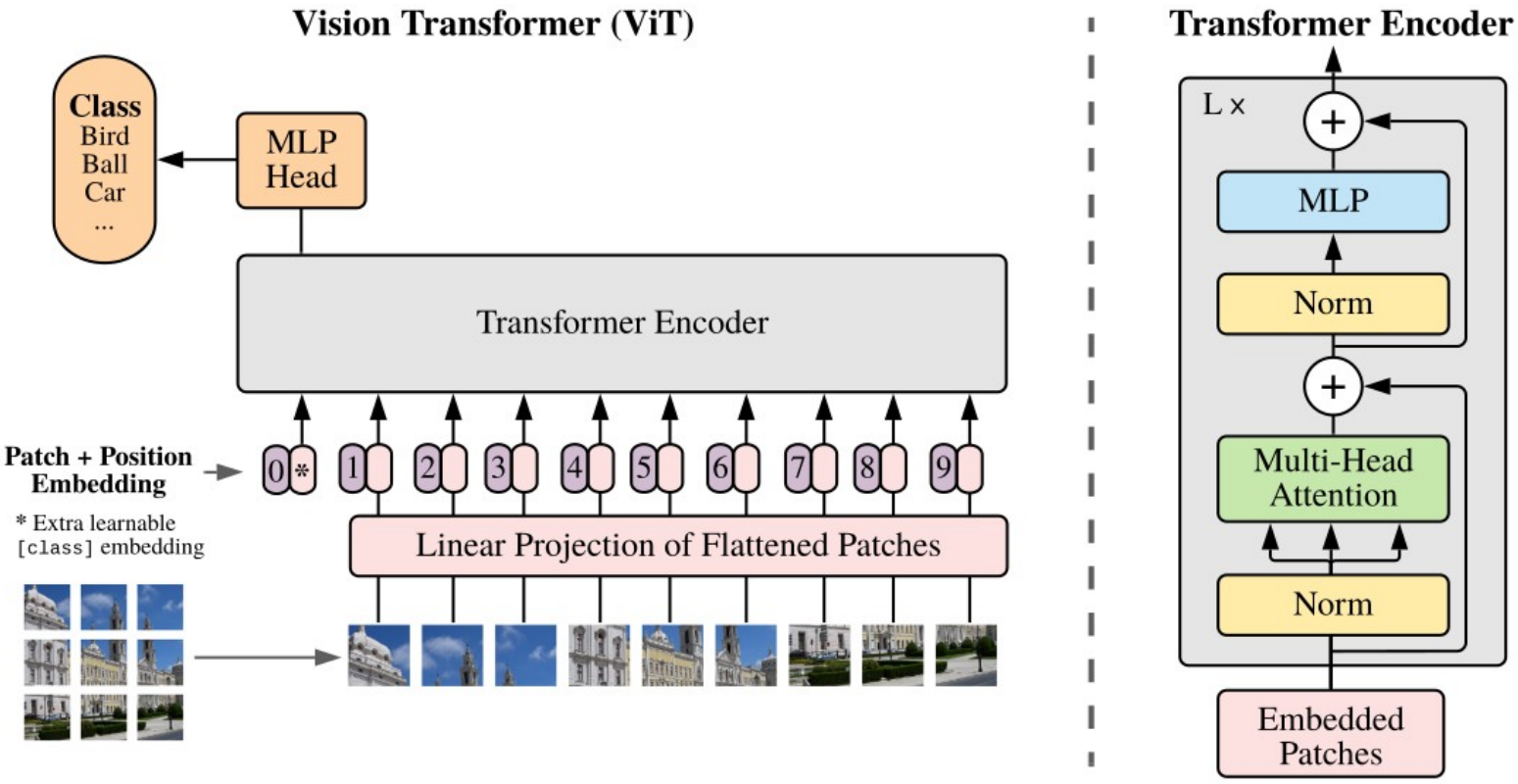


Remarque : Il existe d’autres façons d’encoder la position (ex: “Fourier Features” ou RoPE).

IV) Application à des images

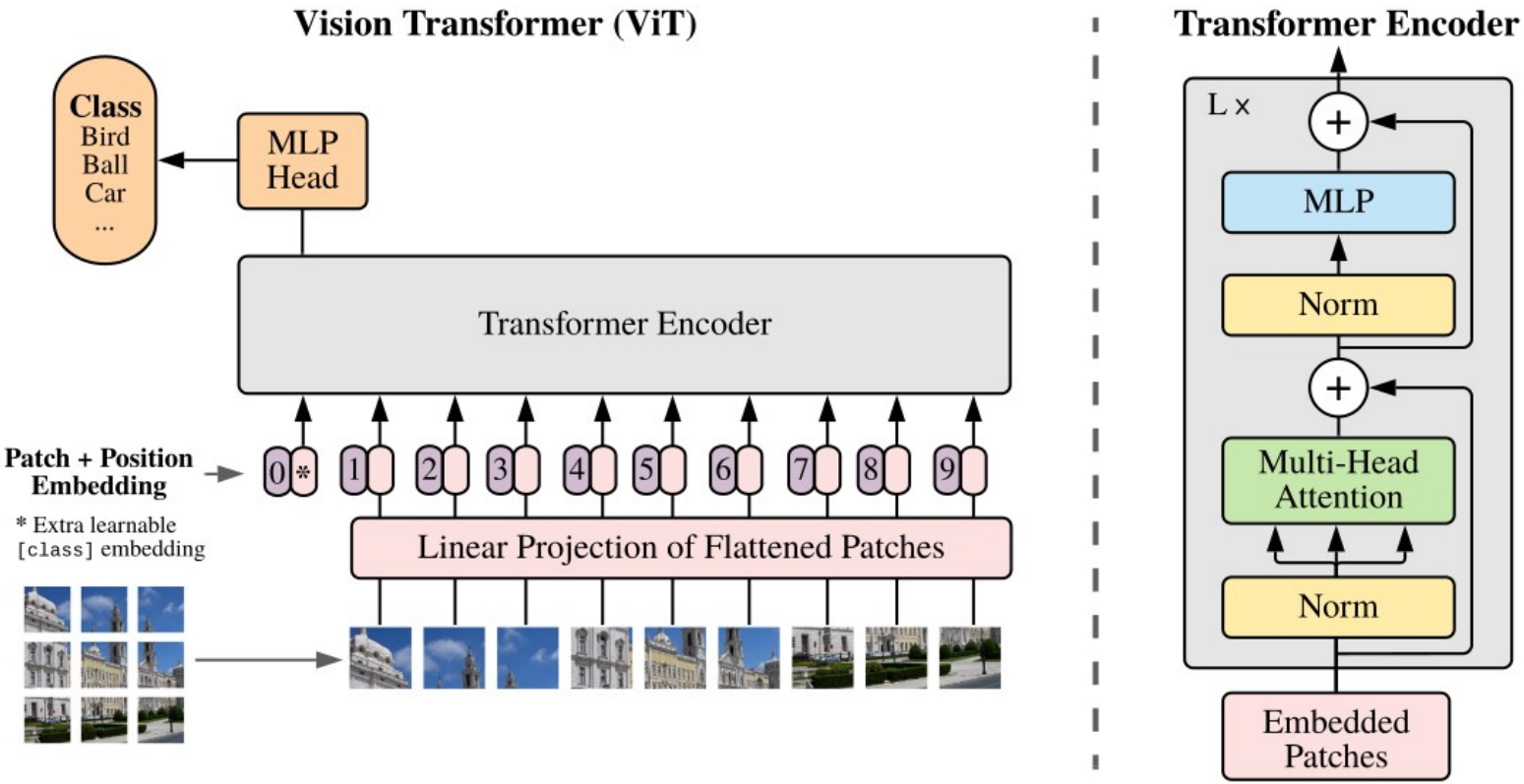
IV)

“An image is worth 16x16 words”, ICLR 2021



IV)

“An image is worth 16x16 words”, ICLR 2021



Architecture ViT très utilisée de nos jours.

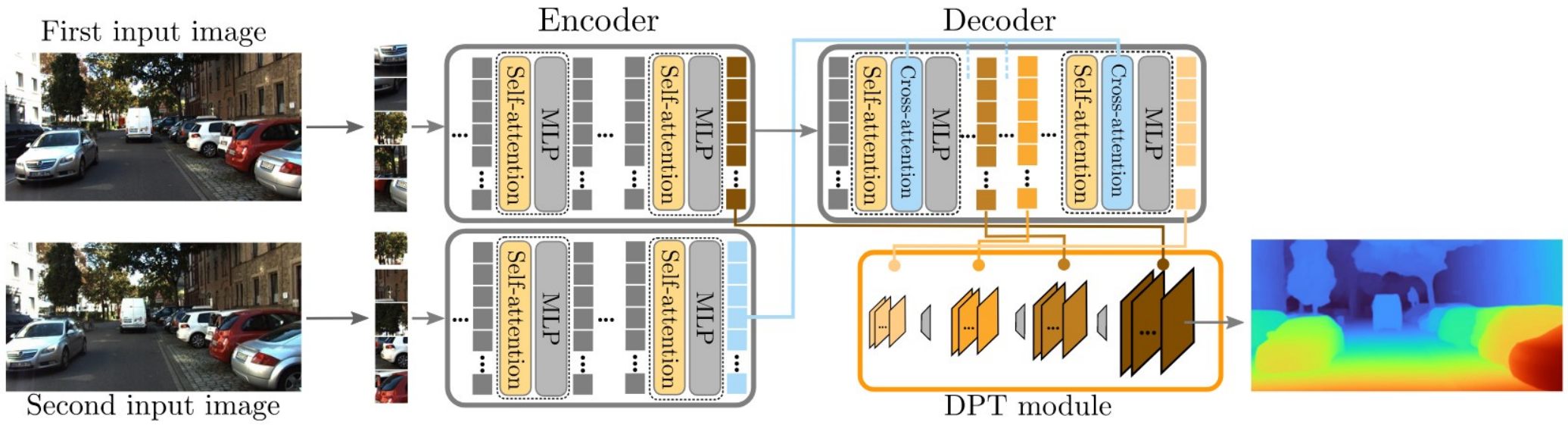
IV)

Architectures DINOv2 et DINOv3

	DINOv2	DINOv3
Backbone	ViT-giant	ViT-7B
#Params	1.1B	6.7B
#Blocks	40	40
Patch Size	14	16
Pos. Embeddings	Learnable	RoPE
Registers	4	4
Embed. Dim.	1536	4096
FFN Type	SwiGLU	SwiGLU
FFN Hidden Dim.	4096	8192
Attn. Heads	24	32
Attn. Heads Dim.	64	128

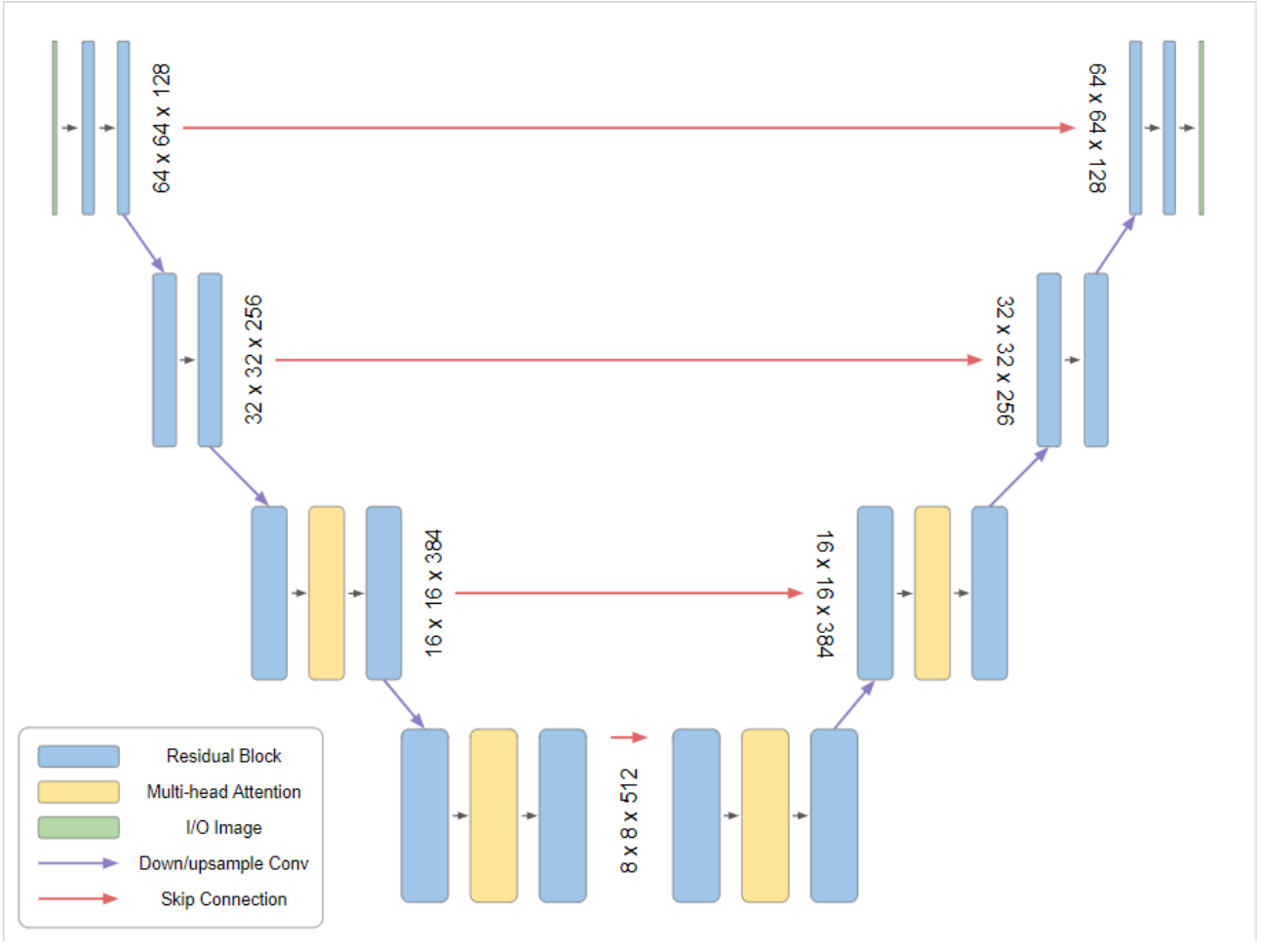
IV)

“CroCo v2: Improved Cross-view Completion Pre-training for Stereo Matching and Optical Flow”, CVPR 2023



IV)

U-Net avec des blocs d'attention



Architecture très utilisée par les méthodes de diffusion (prochain cours)

V) Limites

v)

Limites des couches d'attention à softmax

$\mathtt{M} = \mathtt{XQ}(\mathtt{YK})^\top$: $N_x \times N_y$ $\quad$ $\mathtt{S} = \mathrm{softmax}(\mathtt{M}, \mathrm{dim}{=}1)$: $N_x \times N_y$

Problème : Inapplicable pour des ensembles de grandes tailles.

V)

Limites des couches d'attention à softmax

$\mathtt{M} = \mathtt{XQ}(\mathtt{YK})^\top$: $N_x \times N_y$ $\quad\quad$ $\mathtt{S} = \mathrm{softmax}(\mathtt{M}, \mathrm{dim}{=}1)$: $N_x \times N_y$

Problème : Inapplicable pour des ensembles de grandes tailles.

Solutions :

- Appliquer des couches d'attention à softmax en cherchant à réduire Nx et/ou Ny
  → **ViT ou U-Net avec attention**
- Modifier la couche d'attention softmax

v)

Exemple de modification de la couche d'attention softmax : couche d'attention linéaire

"Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention", ICML 2020

$$\mathbf{x}' = \mathbf{x} + \left(\sum_{j=1}^{N_y} \frac{\phi(\mathbf{x}\mathtt{Q})\phi(\mathbf{y}_j\mathtt{K})^\top}{\sum_{k=1}^{N_y} \phi(\mathbf{x}\mathtt{Q})\phi(\mathbf{y}_k\mathtt{K})^\top} \mathbf{y}_j \right) \mathtt{V}$$

Remplacement du noyau exponentiel par un noyau linéaire

Se simplifie en

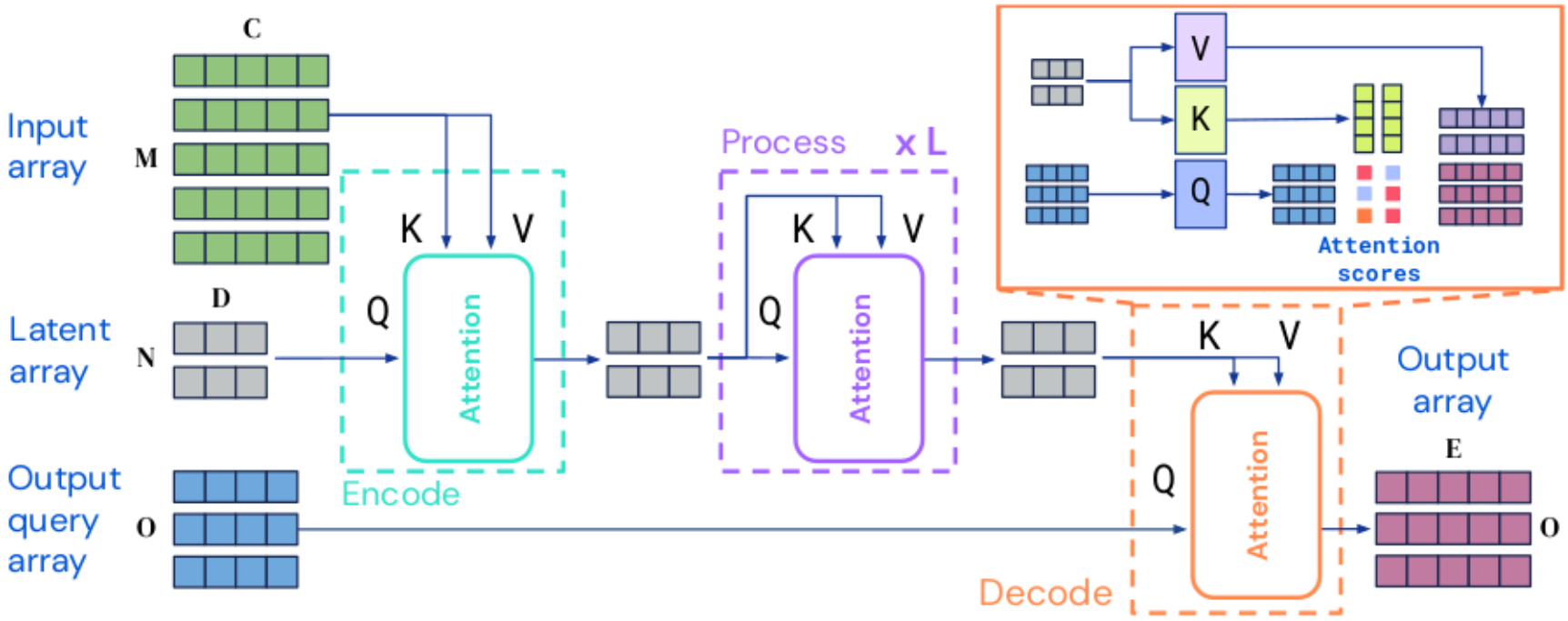
$$\mathbf{x}' = \mathbf{x} + \phi(\mathbf{x}\mathtt{Q}) \frac{\boxed{\sum_{j=1}^{N_y} \phi(\mathbf{y}_j\mathtt{K})^\top \mathbf{y}_j \mathtt{V}}}{\phi(\mathbf{x}\mathtt{Q}) \boxed{\sum_{k=1}^{N_y} \phi(\mathbf{y}_k\mathtt{K})^\top}}$$

Indépendant de x, plus besoin de calculer ni stocker explicitement les matrices M et S

v)

Exemple de réduction de Nx et/ou Ny : PerceiverIO

"Perceiver IO: A General Architecture for Structured Inputs & Outputs." arXiv, 2021



Annexe : Hallucinations

Prompt : “What was the title of Adam Kalai’s dissertation?”

ChatGPT: (GPT-4o)	Adam Tauman Kalai’s Ph.D. dissertation (completed in 2002 at CMU) is entitled: “Boosting, Online Algorithms, and Other Topics in Machine Learning.”
DeepSeek:	“Algebraic Methods in Interactive Machine Learning”. . . at Harvard University in 2005.
Llama:	“Efficient Algorithms for Learning and Playing Games”. . . in 2007 at MIT.

Le vrai titre est “Probabilistic and on-line methods in machine learning.”

→ Les LLM ne sont pas fiables ! Il faut vérifier les réponses.

“Why Language Models Hallucinate”, OpenAI 2025

Annexe : In-Context Learning

Prompt

English: Hello
French: Bonjour

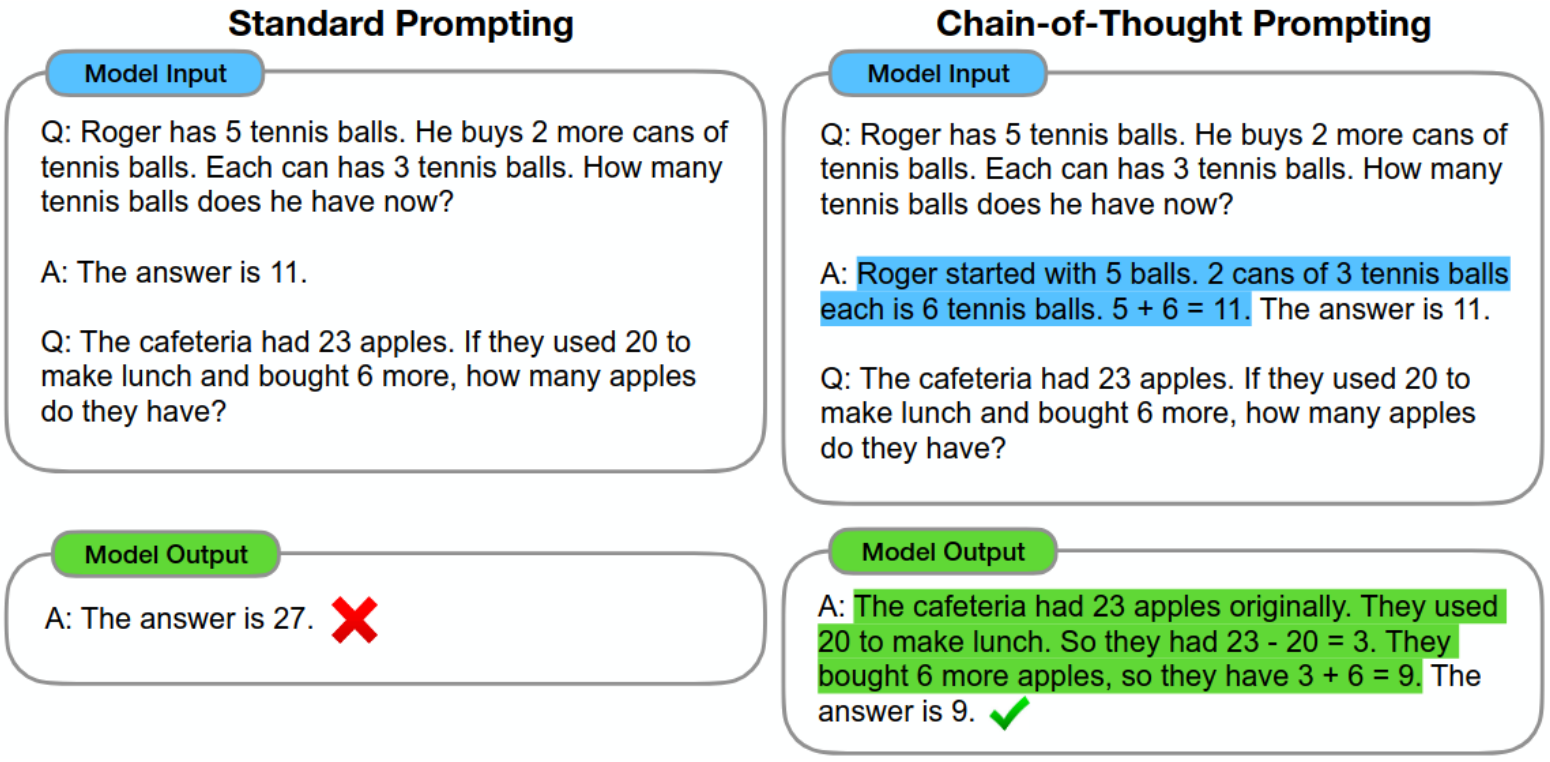
English: Thank you
French: Merci

English: How are you?
French:

→ Les couches d'attention permettent "d'apprendre à faire une tâche", ici traduire, sans faire un apprentissage (c'est-à-dire sans modifier les paramètres du réseau), simplement à partir du prompt.

"Transformers learn in-context by gradient descent", ICML 2023

Annexe : Chain-of-thought



"Chain-of-Thought Prompting Elicits Reasoning in Large Language Models", NeurIPS 2022

Annexe : Scaling laws

En pratique :

- Le nombre de GPU disponible est connu
- La date à laquelle on voudrait déployer un modèle est fixée

→ La quantité de calculs (FLOPs) qu'on va pouvoir faire pendant l'entraînementest connue et on voudrait savoir s'il vaut mieux :

1) Agrandir le réseau et donc voir moins de données pendant l'entraînement
2) Garder un réseau petit et donc voir plus de données pendant l'entraînement

"Training Compute-Optimal Large Language Models", NeurIPS 2022

Annexe : Scaling laws (suite)

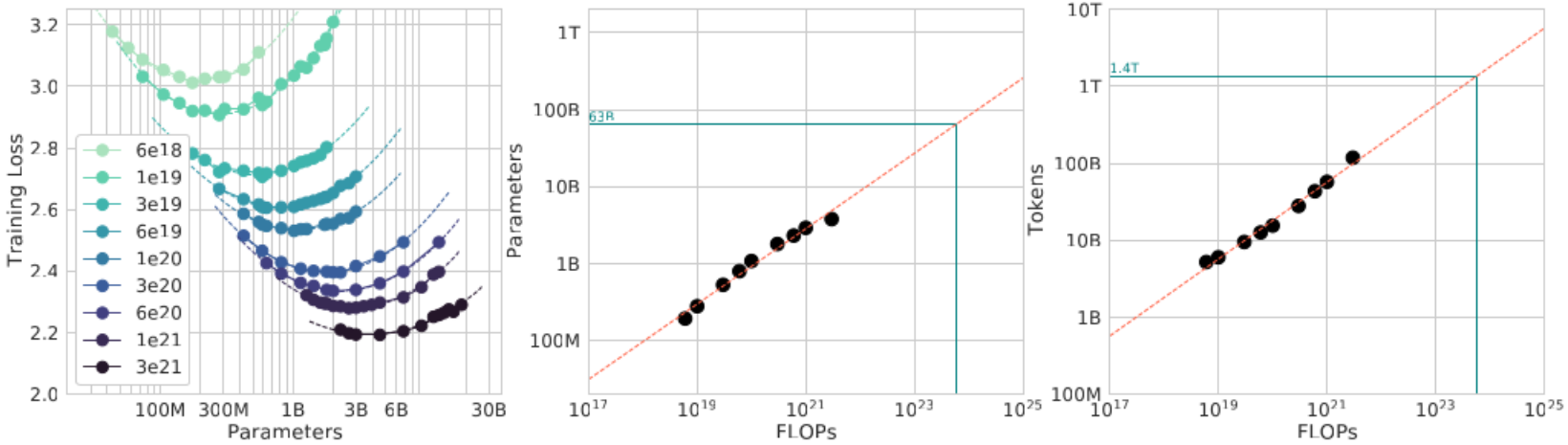
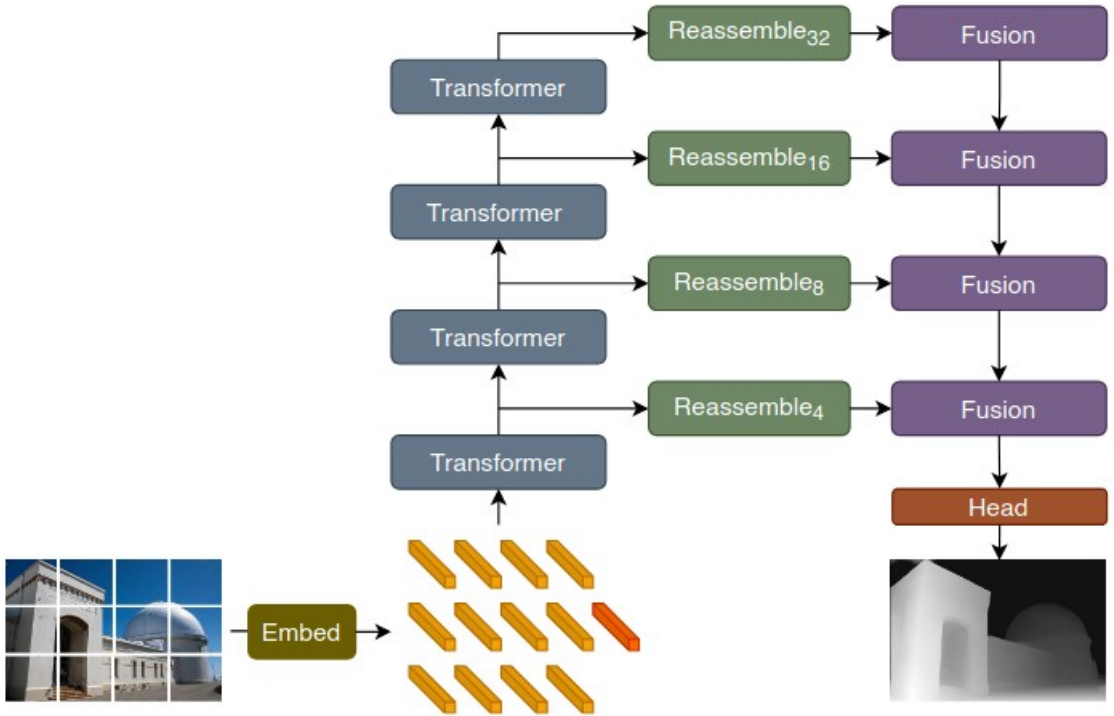


Figure 3 | **IsoFLOP curves.** For various model sizes, we choose the number of training tokens such that the final FLOPs is a constant. The cosine cycle length is set to match the target FLOP count. We find a clear valley in loss, meaning that for a given FLOP budget there is an optimal model to train (**left**). Using the location of these valleys, we project optimal model size and number of tokens for larger models (**center** and **right**). In green, we show the estimated number of parameters and tokens for an *optimal* model trained with the compute budget of *Gopher*.

"Training Compute-Optimal Large Language Models", NeurIPS 2022

Annexe : Tête DPT (“Dense Prediction Transformer”)



“Vision Transformers for Dense Prediction”, CVPR 2021