

Traitement et Analyse d'Images

Modèles d'attention Transformers

Quelles applications pour les modèles d'attention ?

▶ NLP: Natural Language Processing ou *traitement de langage naturel*

Permettre aux machines de comprendre, interpréter et générer un langage naturel de manière efficace

▶ Développé pour des tâches telles que

- Traduction automatique
- Compréhension de la parole
- Génération de texte
- Génération de code informatique

▶ Algorithmes connus

- BERT
- GPT 3, 4
- Chat-GPT
- [...]

explain quantum theory to a child in the style of snoop dogg

Yo, little homie, let me break it down for you
Quantum theory, it's a crazy thing, it's true
See, the world is made up of tiny, tiny particles
That are always moving, and changing, and actin' wild

These particles, they don't always behave like we think
Sometimes they're waves, and sometimes they're points, it's a trip
They can be in two places at once, and they can be in two states
It's a mind-blowing concept, but that's the quantum fate

So don't be scared, little homie, it's all good
Quantum theory, it's just the way the world works, it's understood
It's a wild ride, but it's a cool one too
Quantum theory, it's the way the world does what it do.

Quelles applications pour les modèles d'attention ?

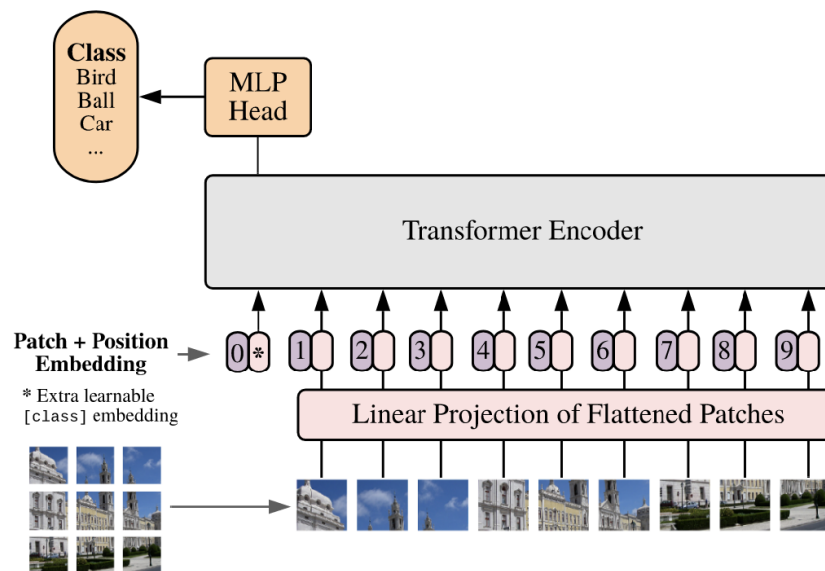
- ▶ Également développé en analyse d'images

Permettre aux machines de comprendre, analyser et générer des images de manière efficace

- ▶ Algorithmes de l'état de l'art

- Classification
- Segmentation
- Segmentation + temps
- Suivi
- Génération d'images

Vision Transformer - 2020



Quelles applications pour les modèles d'attention ?

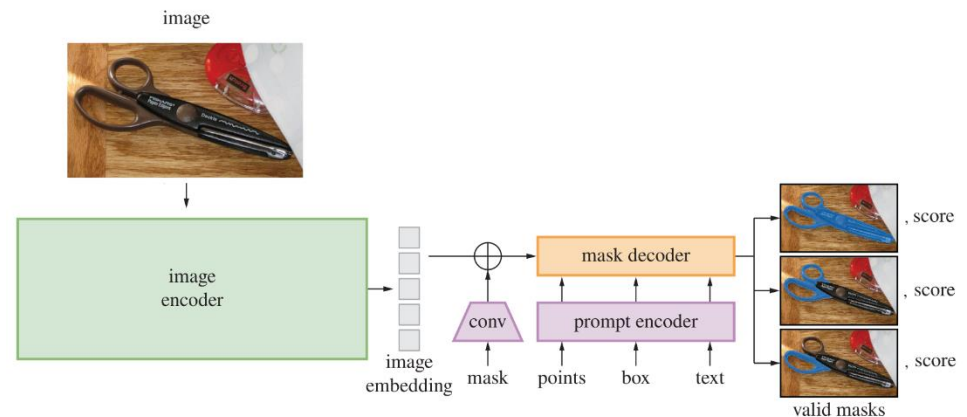
- ▶ Également développé en analyse d'images

Permettre aux machines de comprendre, analyser et générer des images de manière efficace

- ▶ Algorithmes de l'état de l'art

- Classification
- Segmentation
- Segmentation + temps
- Suivi
- Génération d'images

SAM (Segment Anything) - 2024



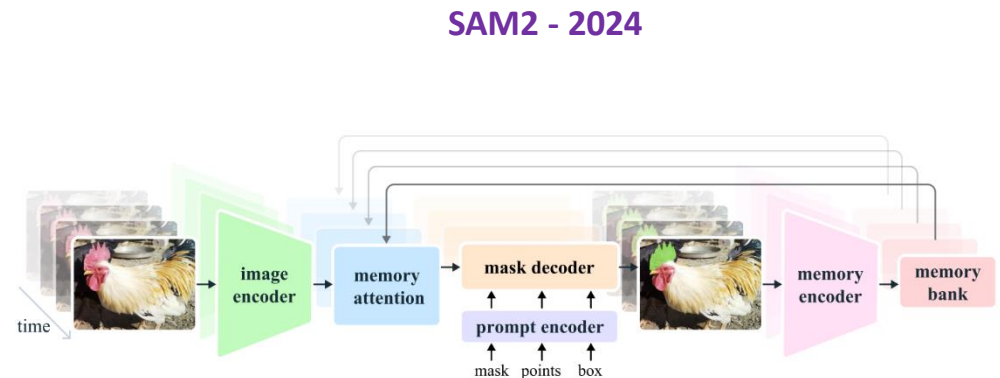
Quelles applications pour les modèles d'attention ?

- ▶ Également développé en analyse d'images

Permettre aux machines de comprendre, analyser et générer des images de manière efficace

- ▶ Algorithmes de l'état de l'art

- Classification
- Segmentation
- Segmentation + temps
- Suivi
- Génération d'images



Quelles applications pour les modèles d'attention ?

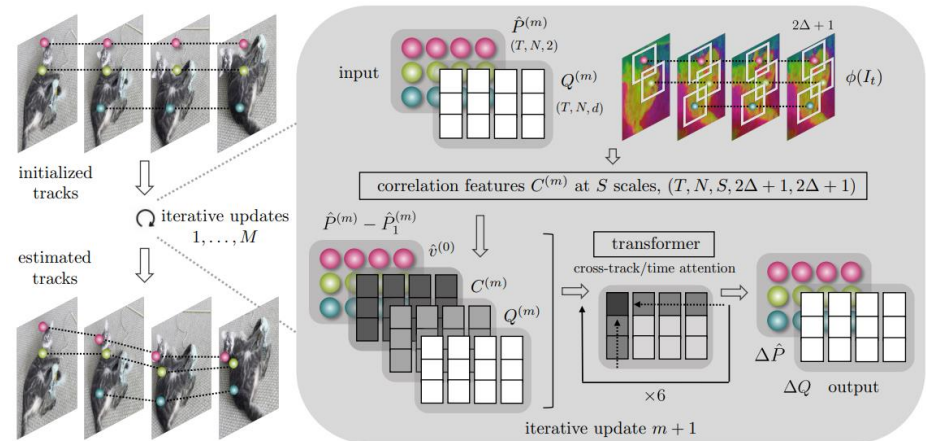
- ▶ Également développé en analyse d'images

Permettre aux machines de comprendre, analyser et générer des images de manière efficace

- ▶ Algorithmes de l'état de l'art

- Classification
- Segmentation
- Segmentation + temps
- Suivi
- Génération d'images

CoTracker - 2024



Quelles applications pour les modèles d'attention ?

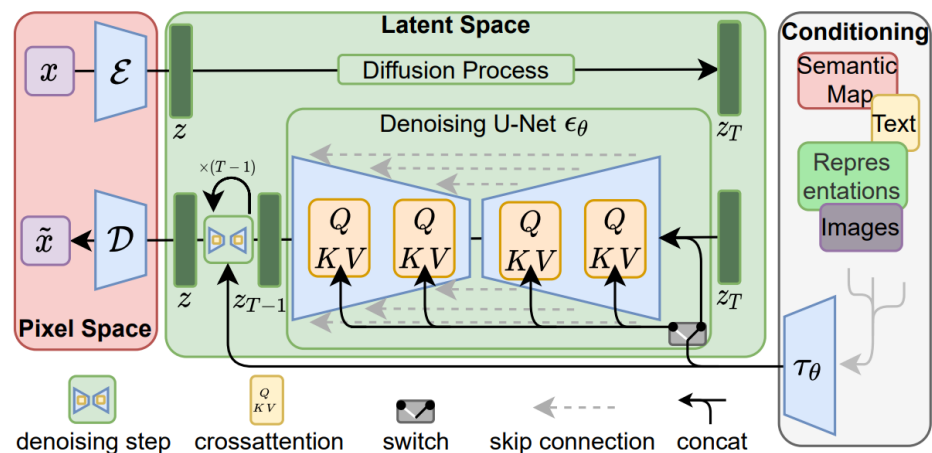
- ▶ Également développé en analyse d'images

Permettre aux machines de comprendre, analyser et générer des images de manière efficace

- ▶ Algorithmes de l'état de l'art

- Classification
- Segmentation
- Segmentation + temps
- Suivi
- Génération d'images

Latent diffusion model - 2023



Les transformers

Tokens

- ▶ Les données d'entrée sont structurées sous forme de *tokens*

- ➔ Texte: token = mots d'une phrase
- ➔ Image: token = patches d'une image

Texte

« Hello, I am olivier »

Procédure de tokenisation

« Hello », « I », « am », « olivier »

« Hello » $\Rightarrow x_i \in \mathbb{R}^t$

0	1	0	0	0
---	---	---	---	---

Image



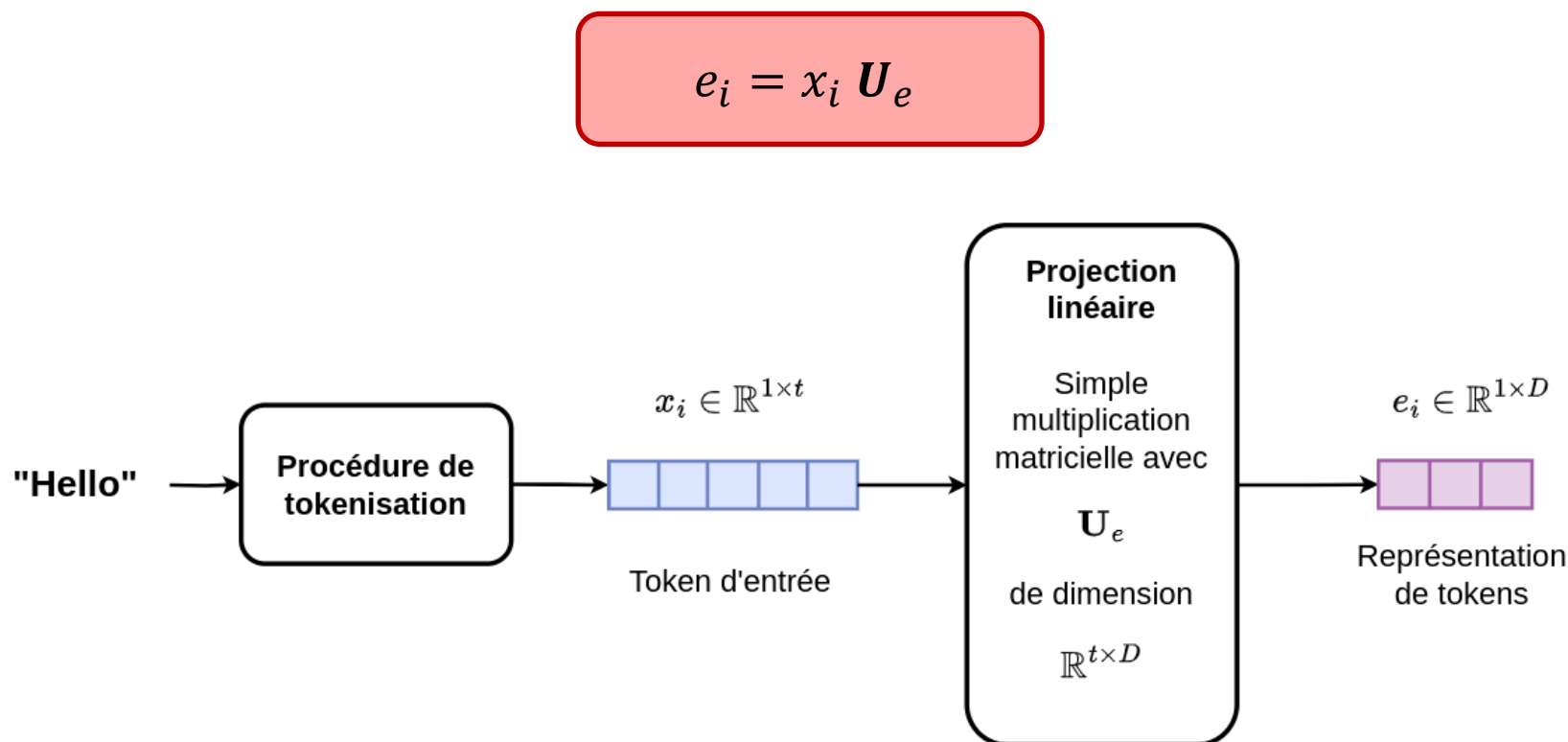
Procédure de tokenisation



$\Rightarrow x_i \in \mathbb{R}^{P^2C}$

P largeur du patch, $C = 3$ si image couleur

- ▶ Création d'une représentation (ou *embedding*) des tokens
 - Simple projection linéaire
 - Multiplication par une matrice de représentation U_e à apprendre

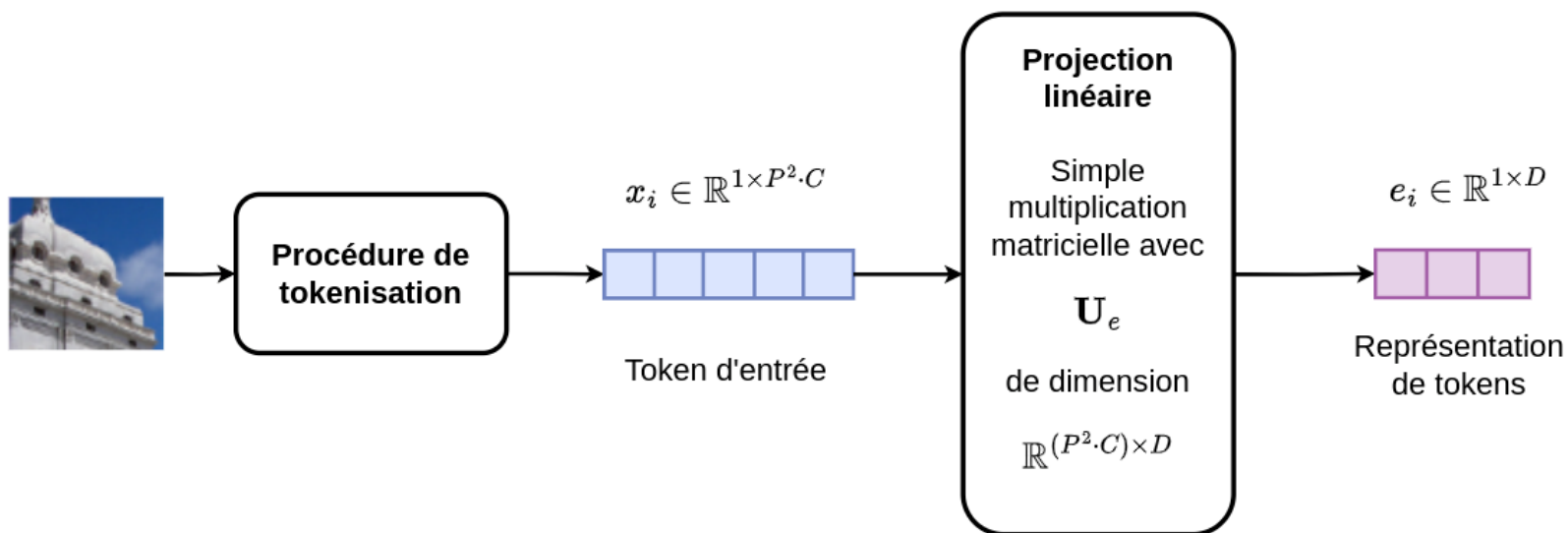


► Création d'une représentation (ou *embedding*) des tokens

→ Simple projection linéaire

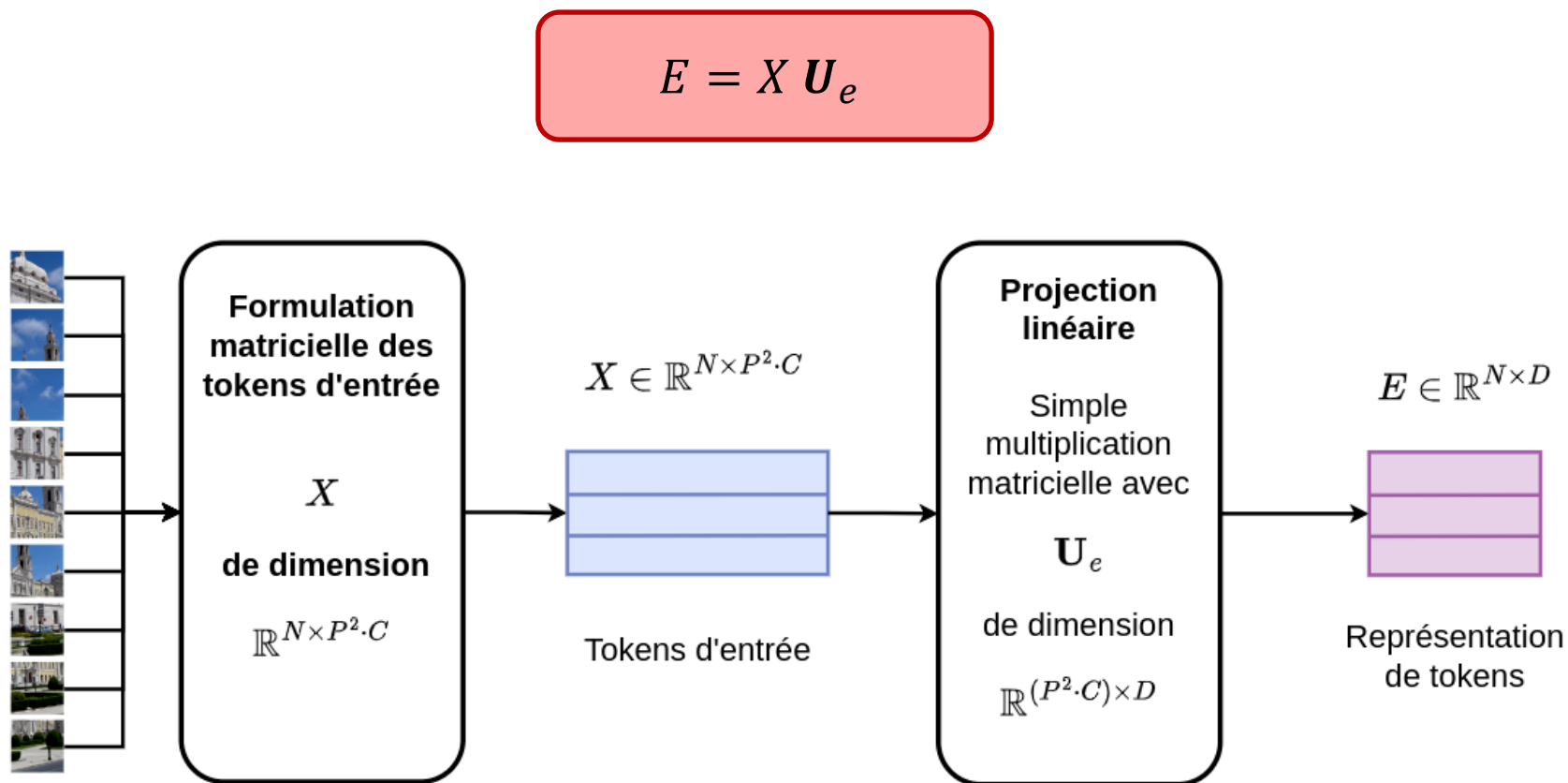
→ Multiplication par une matrice de représentation U_e à apprendre

$$e_i = x_i U_e$$



- ▶ Apprentissage d'une matrice de représentation U_e commune à chaque token

→ Formulation matricielle



- ▶ Codage de position
 - Phrase / image: ensemble de tokens indépendants
 - Perte d'informations structurelles des données d'entrée
- ▶ Récupération de la structure : codage positionnel (PE: positional embedding)
 - Correspondance entre la position du token t et un vecteur $p_t \in \mathbb{R}^{1 \times D}$
 - Codage classique: fonction sinusoïdale

$$p_t \in \mathbb{R}^{1 \times D}$$

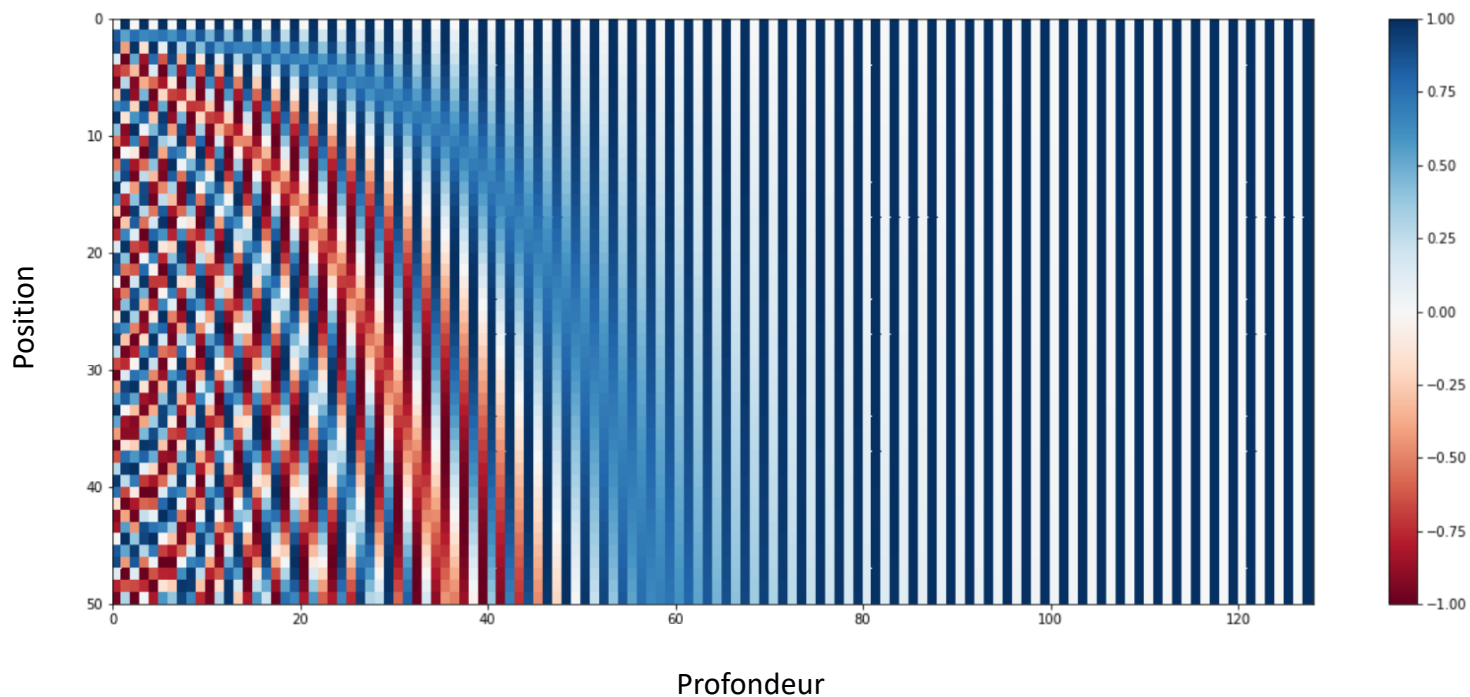
$$p_t = [\sin(\omega_1 t), \cos(\omega_1 t), \dots, \sin(\omega_{D/2} t), \cos(\omega_{D/2} t)]$$

$$\omega_k = \frac{1}{10000^{2k/D}}$$

► Codage de position sinusoidal

→ Unique vecteur p_t pour chaque position t

→ $p_t(i) \in [-1,1]$: normalisation intrinsèque des valeurs

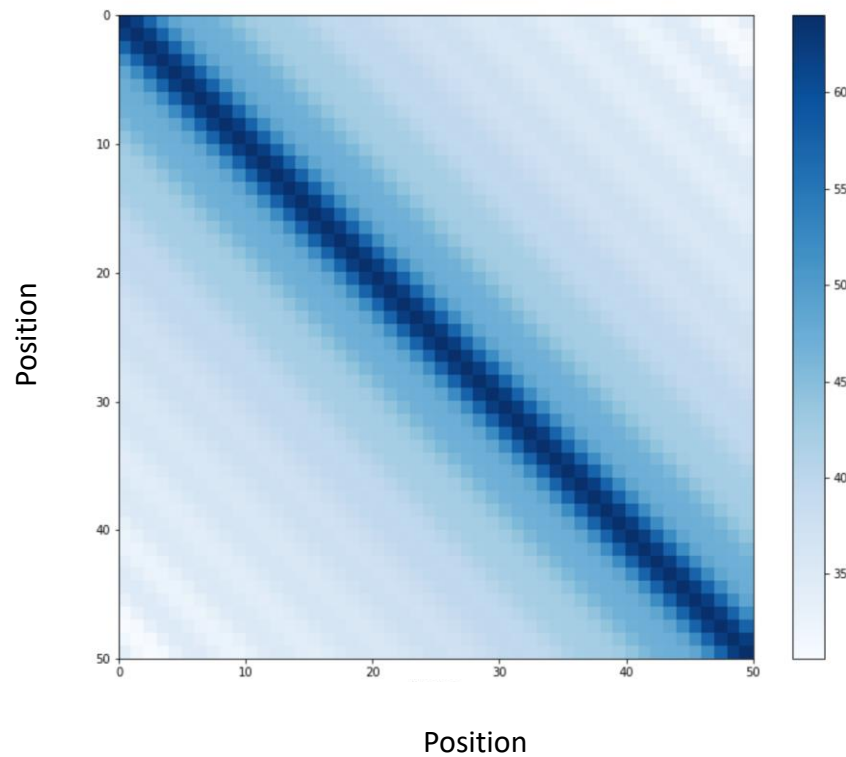


Nombre de token = 50, dimension D de chaque token = 128

► Codage de position sinusoidal

→ Modélisation intrinsèque de la position relative des tokens

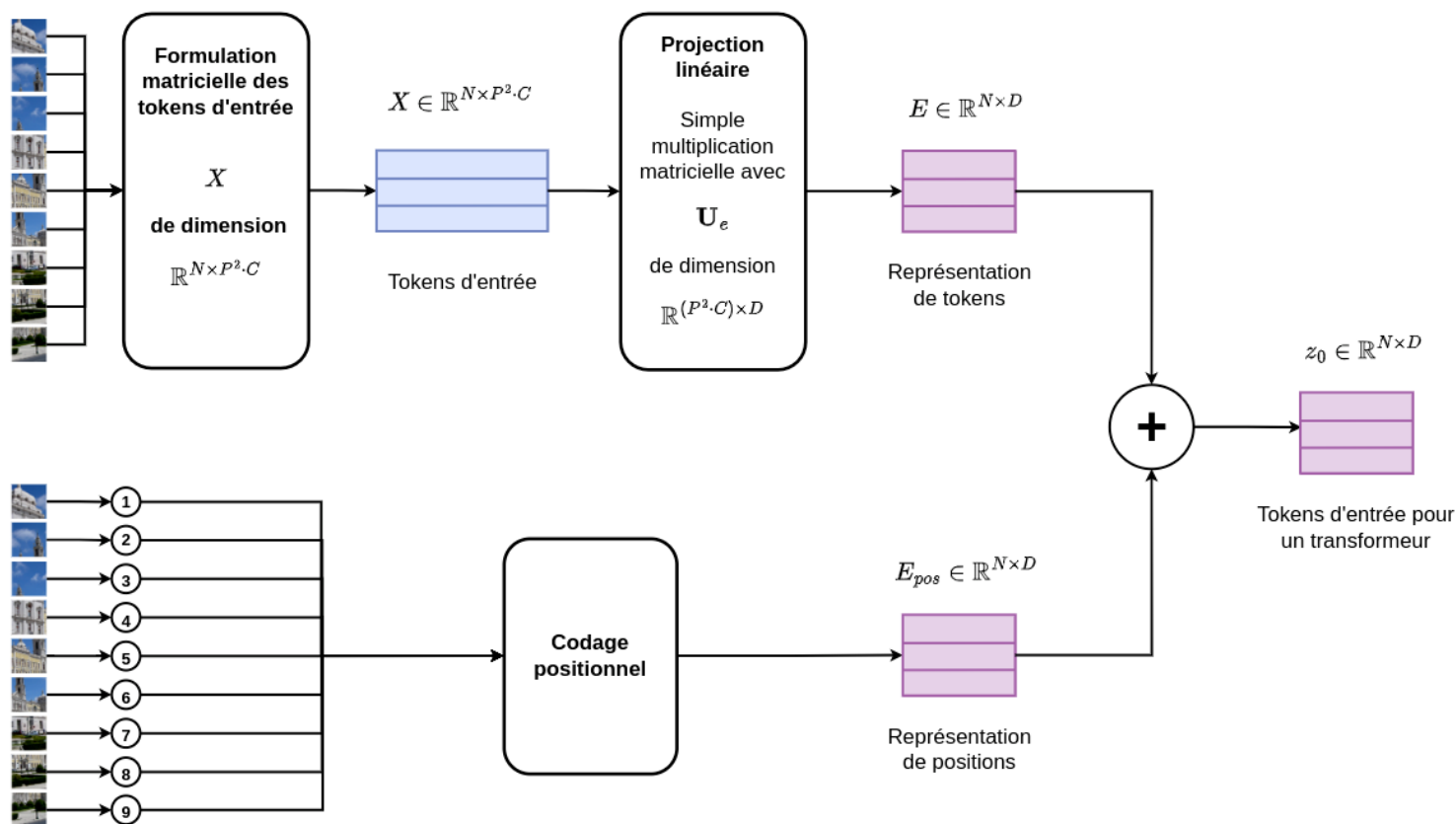
→ Matrice de similarité de position: $K = P \cdot P^t$



► Représentation finale

→ Tokens finaux = somme des représentations de tokens et de positions

→ Seule la matrice U_e est à apprendre pour cette phase



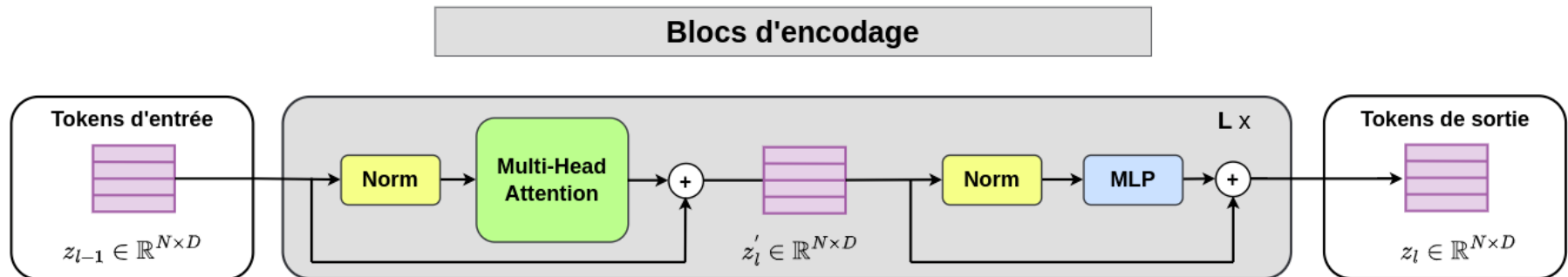
Les transformers

Les blocs d'encodage

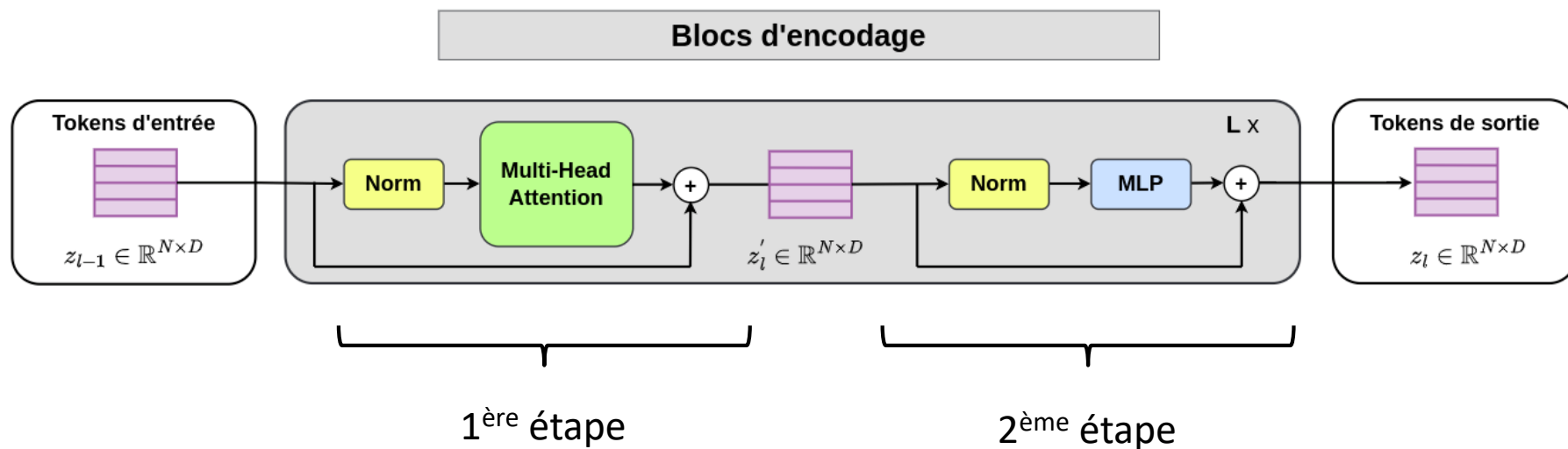
► Encodeur

→ Correspond a N blocs d'encodage

- En entrée: Une représentation de tokens
- En sortie: Une nouvelle représentation de tokens adaptée à la cible en cours d'optimisation



Transformer: encodage de l'information

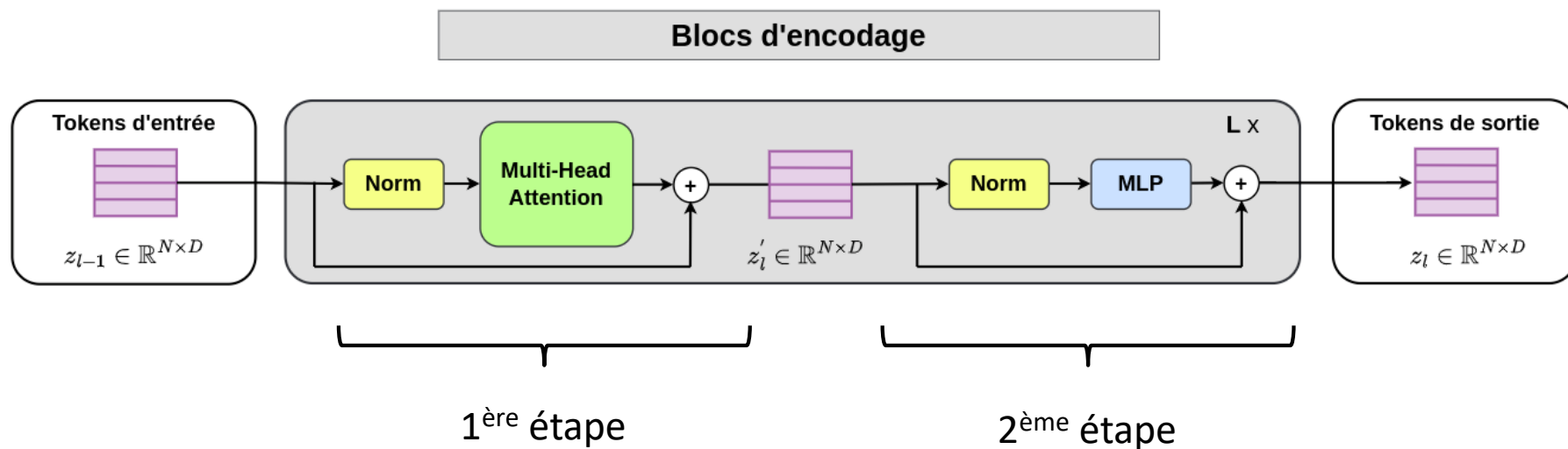


► 1^{ère} étape

- ➔ Calcul de cartes d'attention entre les tokens
- ➔ Connection résiduelle 1) contre les pertes de gradient
2) ne pas oublier la représentation de position

$$z'_l = MHA(LN(z_{l-1})) + z_{l-1}$$

Transformer: encodage de l'information

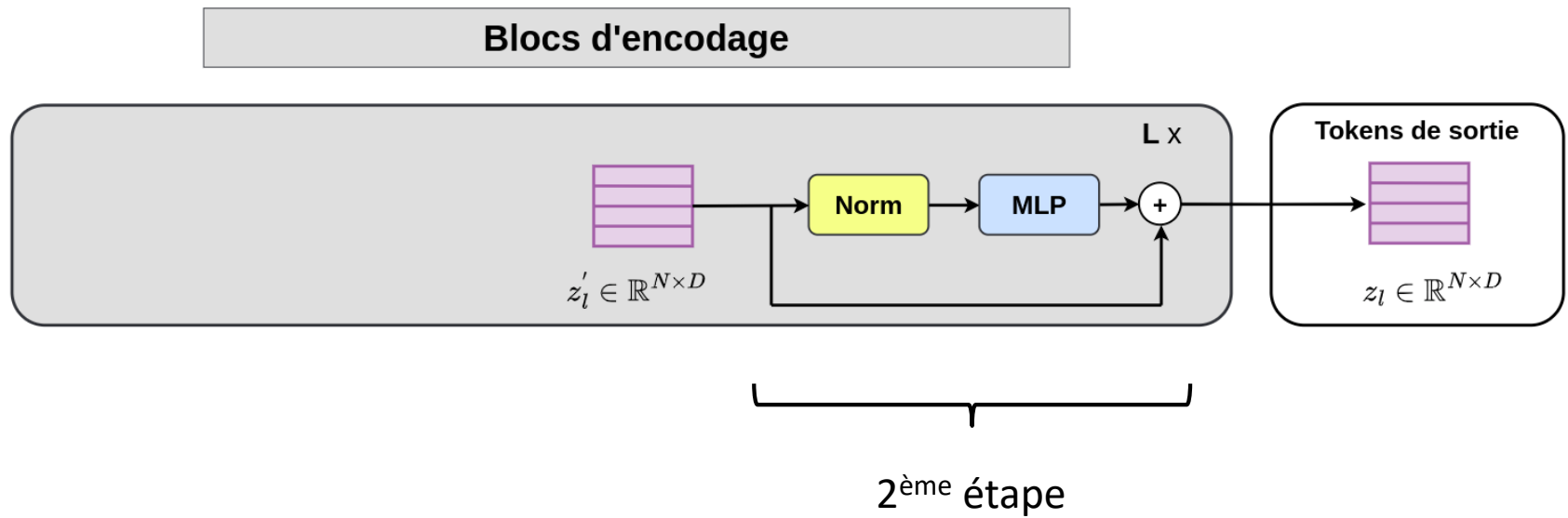


► 2^{ème} étape

- ➔ Introduction de non-linéarités pour générer de l'information pertinente
- ➔ Connection résiduelle 1) contre les pertes de gradient
2) ne pas oublier la représentation de position

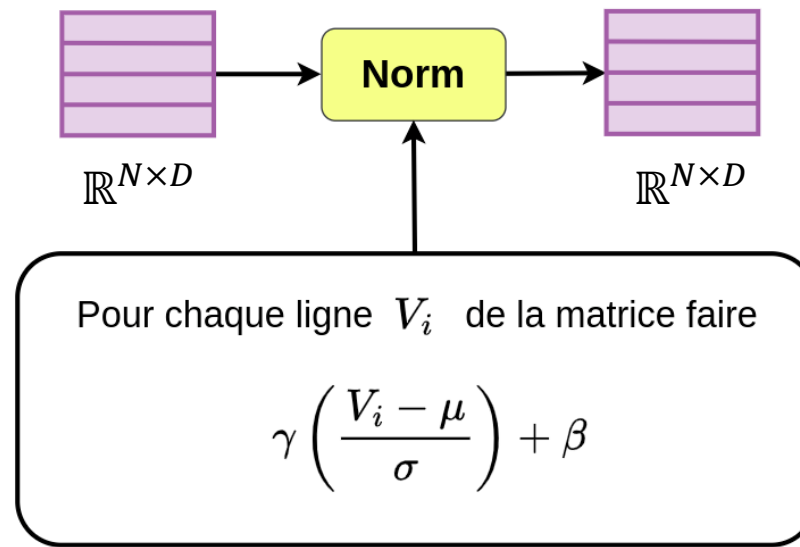
$$z_l = MLP(LN(z'_l)) + z'_l$$

► Zoom sur la 2^{ème} étape



► Normalisation

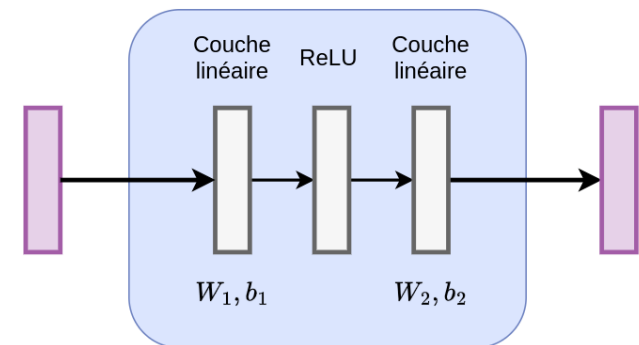
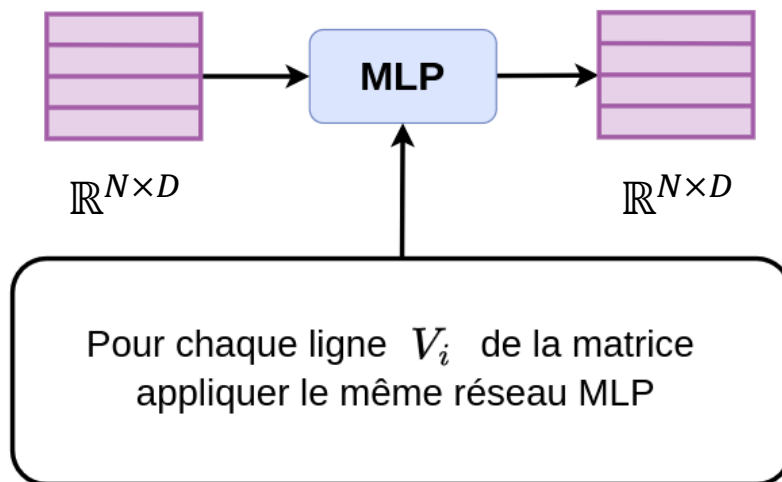
- Contrôle la dynamique des valeurs des tokens avant chaque étape clé
 - μ, σ : Calculés sur l'ensemble des tokens correspondant à une image
 - γ, β : Paramètres à apprendre



► MLP

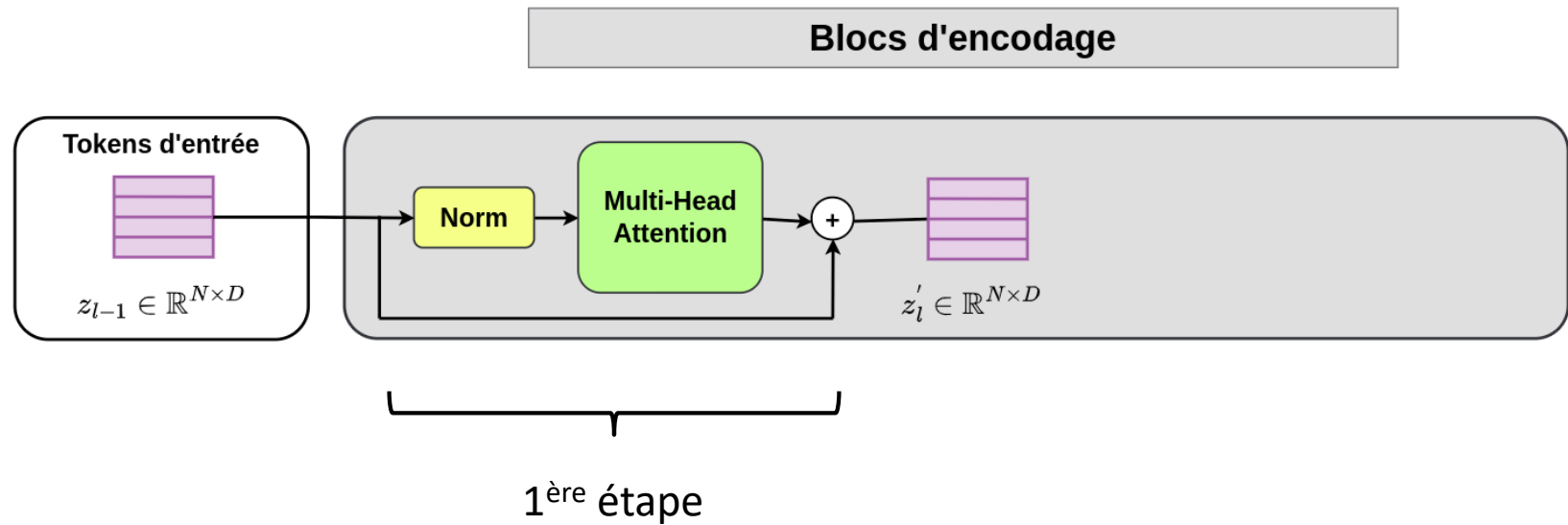
- Introduit de la non-linéarité
- Permet de générer de l'information pertinente

$$z_l^* = LN(z_l')$$
$$MLP(z_l^*) = \max(0, z_l^* W_1 + b_1) W_2 + b_2$$



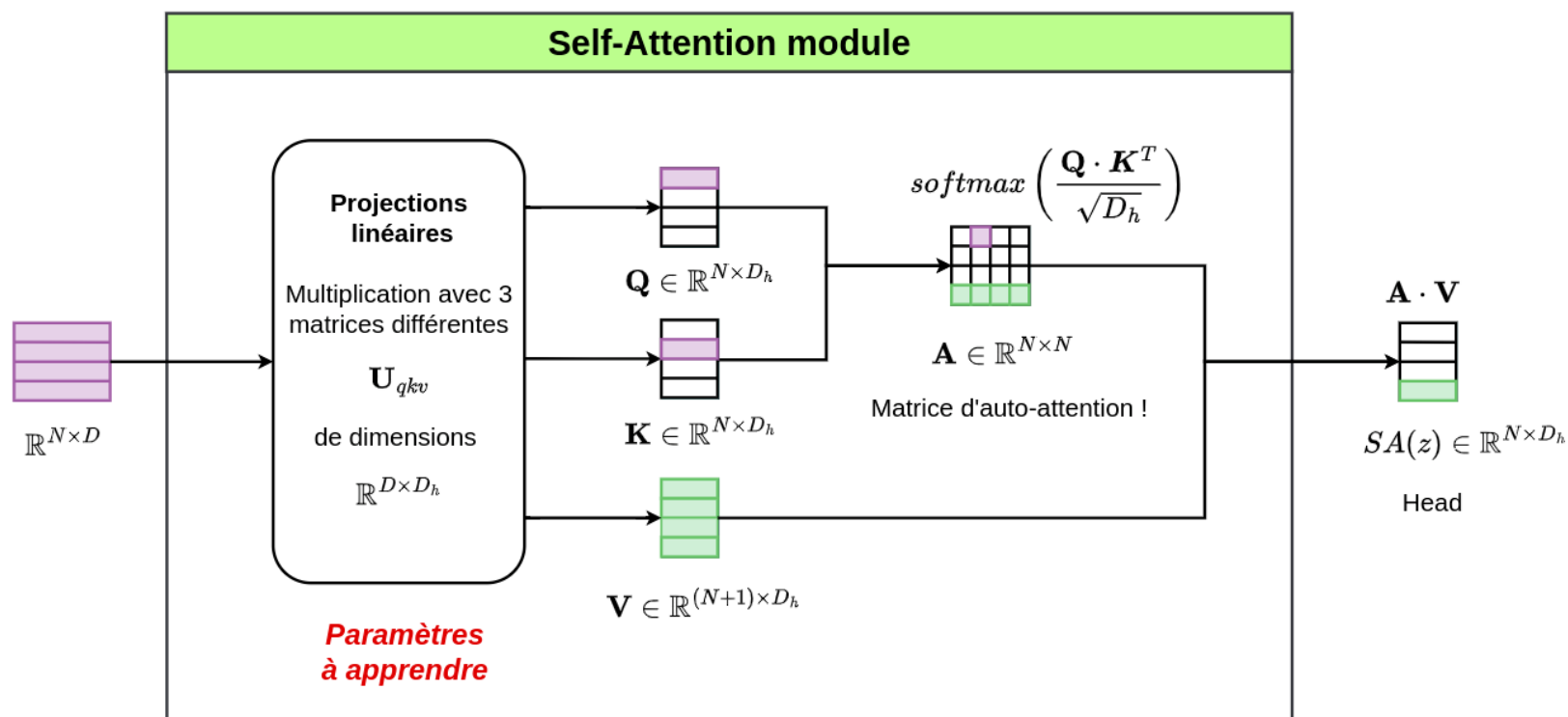
Transformer: encodage de l'information

► Zoom sur la 1^{ère} étape



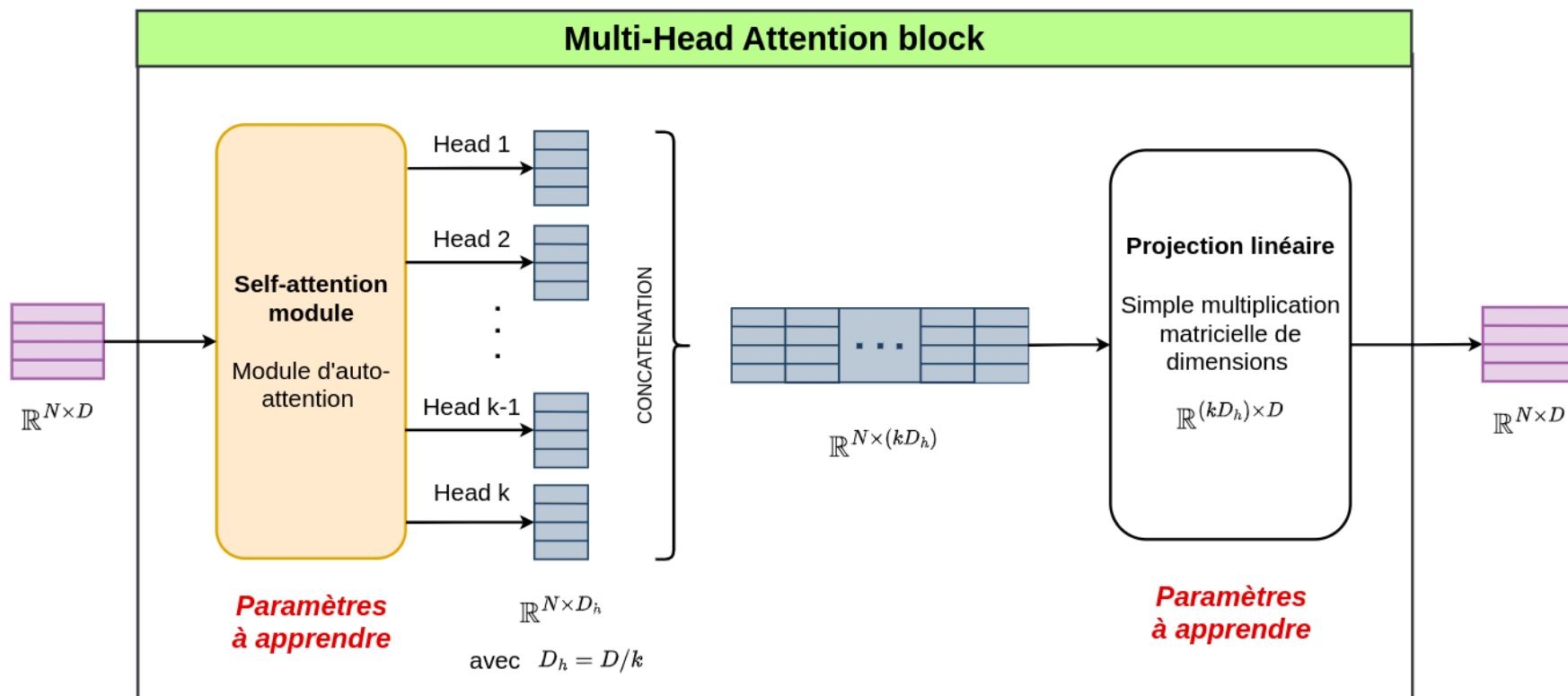
► Self-attention module

- Gestion de cartes d'attentions A au travers de matrices Q (query), K (key), V (values)
- Softmax appliqué par ligne de la matrice A pour normaliser les poids qui vont pondérer les vecteurs lignes de V



Transformer: encodage de l'information

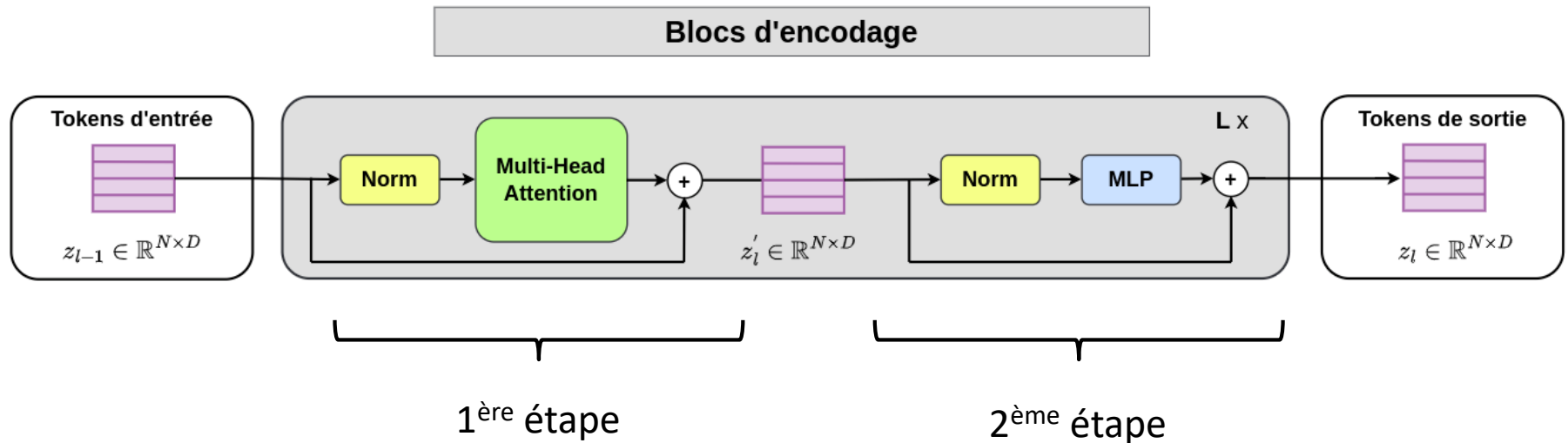
- ▶ Multi-Head Attention: bloc d'attention multi-tête
 - ➔ Génération de k têtes à partir de modules d'auto-attention différents
 - ➔ Equivalent à la notion de *feature maps* dans les *CNN*
 - ➔ Projection linéaire pour mélanger l'information des différentes têtes et revenir aux dimensions de tokens initiaux



Transformer: encodage de l'information

► En résumé

- ➔ 1^{ère} étape: création d'information par attention entre les tokens
- ➔ 2^{ème} étape: génération d'information pertinente par non-linéarité



Les transformers

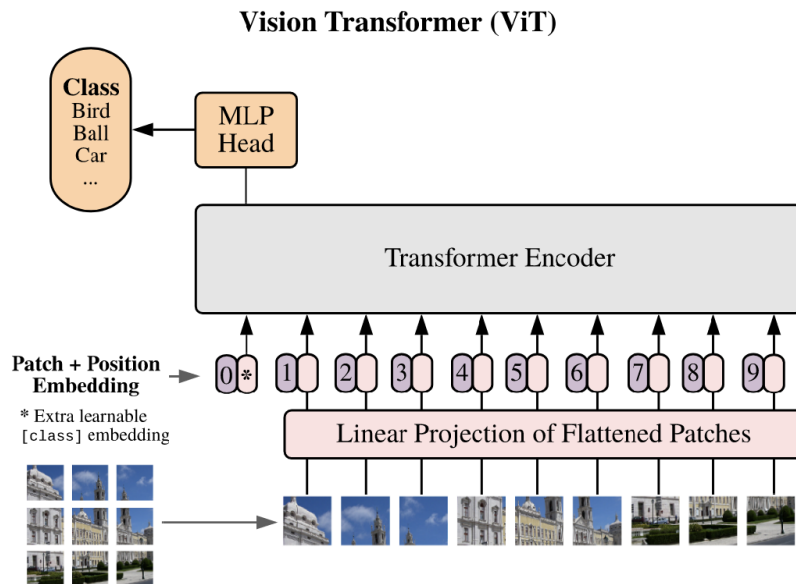
Méthode de classification

Transformer pour la classification

► ViT: algorithme de référence

- Appris sur JFT (300 million d'images)
- Introduction de la notion de *class token*

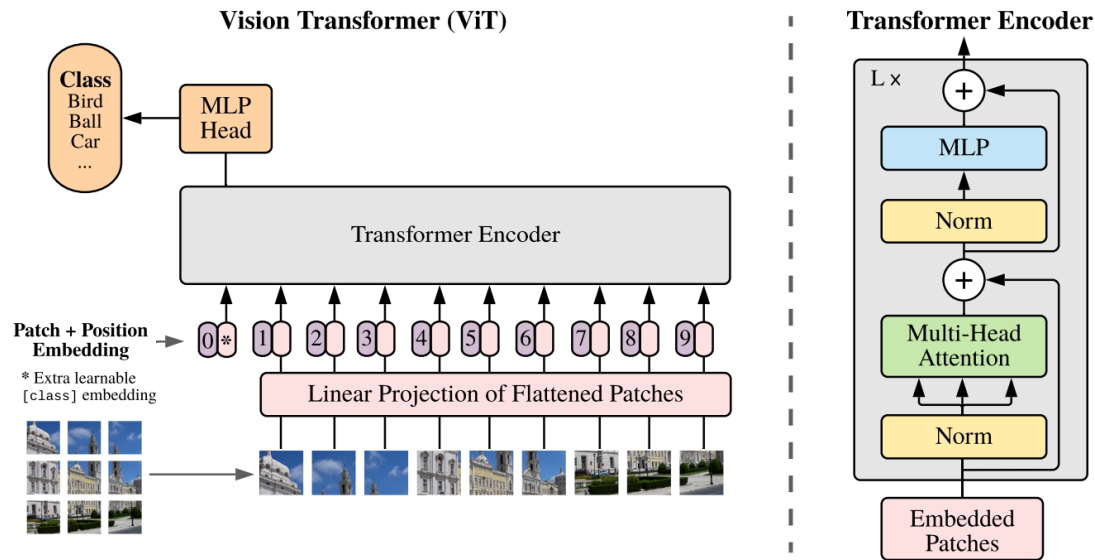
Apprentissage d'une opération de « pooling » vis-à-vis de token visuels



Animation gif

Transformer pour la classification

► ViT: algorithme de référence



Modèles	Couches / blocs	Taille cachée D	Taille MLP	Têtes	Paramètres
ViT-Base	12	768	3072	12	86 M
ViT-Large	24	1024	4096	16	307 M
ViT-Huge	32	1280	5120	16	632 M

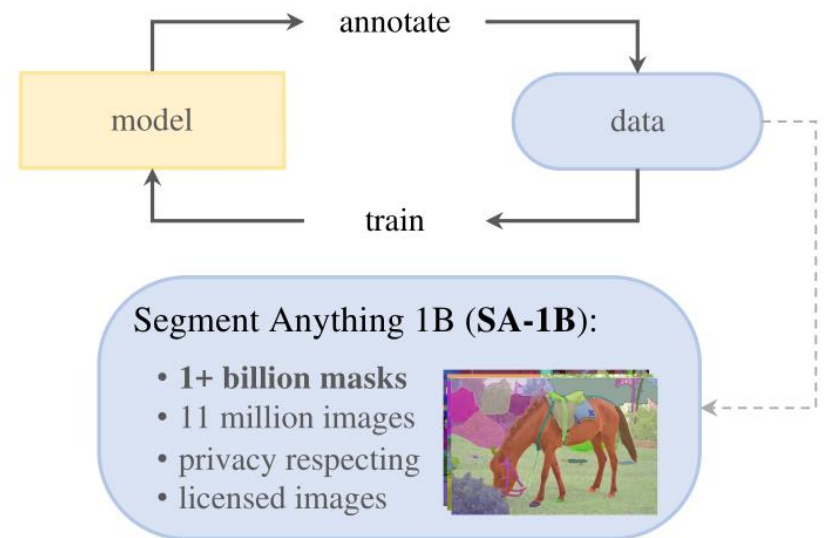
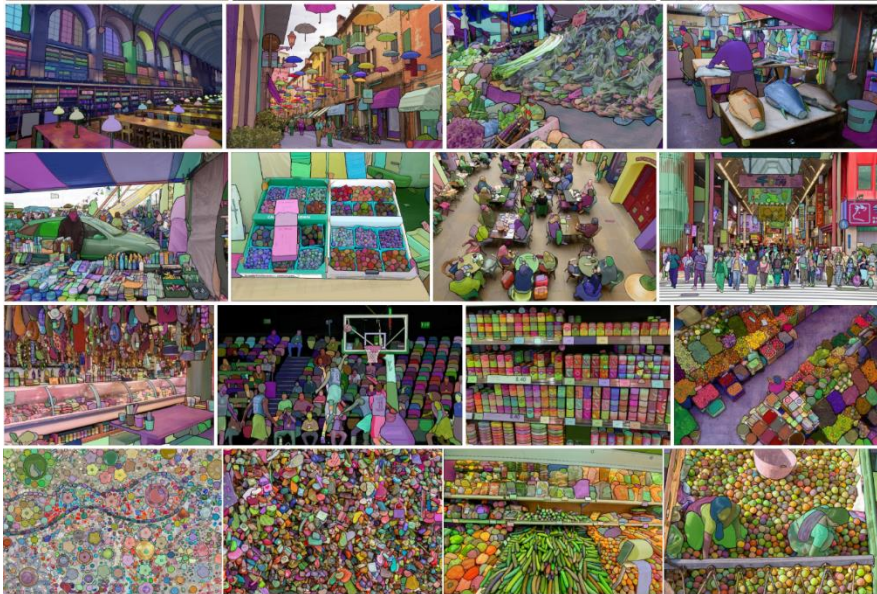
Les transformers

Méthode de segmentation

Transformer pour la segmentation

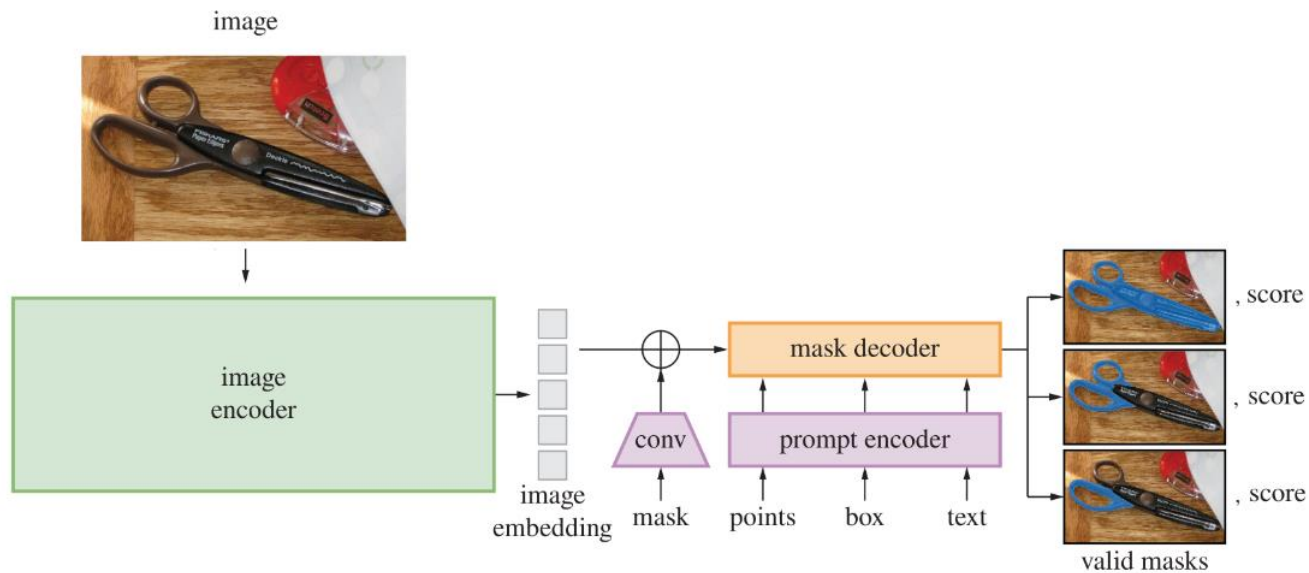
► SAM: Segment Anything

- Appris sur SA-1B (11 million d'images, 1 milliard de masques)
- Images 2D de scènes naturelles, Taille minimale d'un côté: 1500px



► SAM: Segment Anything

- Architecture relativement simple
- Processus de segmentation interactive
- Intègre le principe d'ambiguïté de segmentation en fonction de l'initialisation

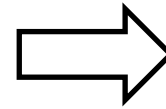
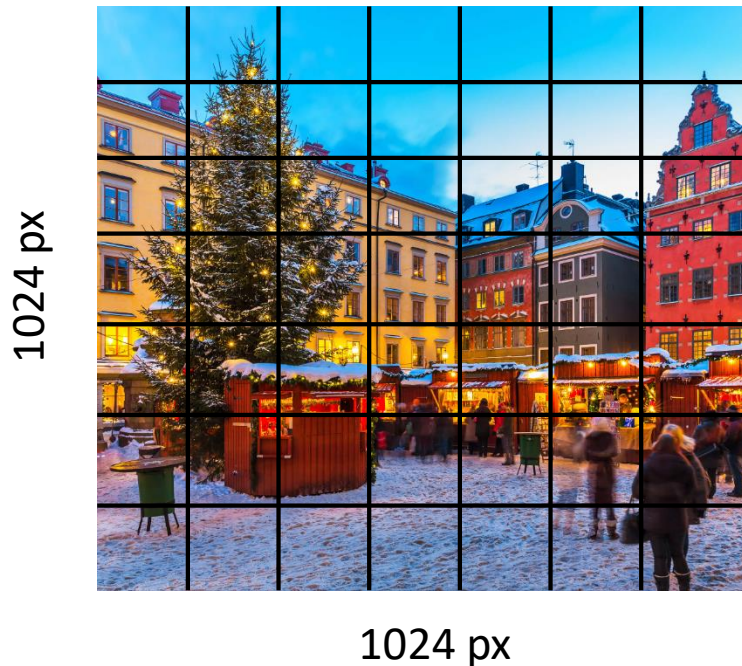


Transformer pour la segmentation

▶ SAM: Segment Anything

→ Image encoder

- ▶ Image d'entrée redimensionnées en 1024 x 1014 px
- ▶ Architecture: Vit-H/16 (token = patch de taille 16 x 16 px)
- ▶ Taille cachée D : 256



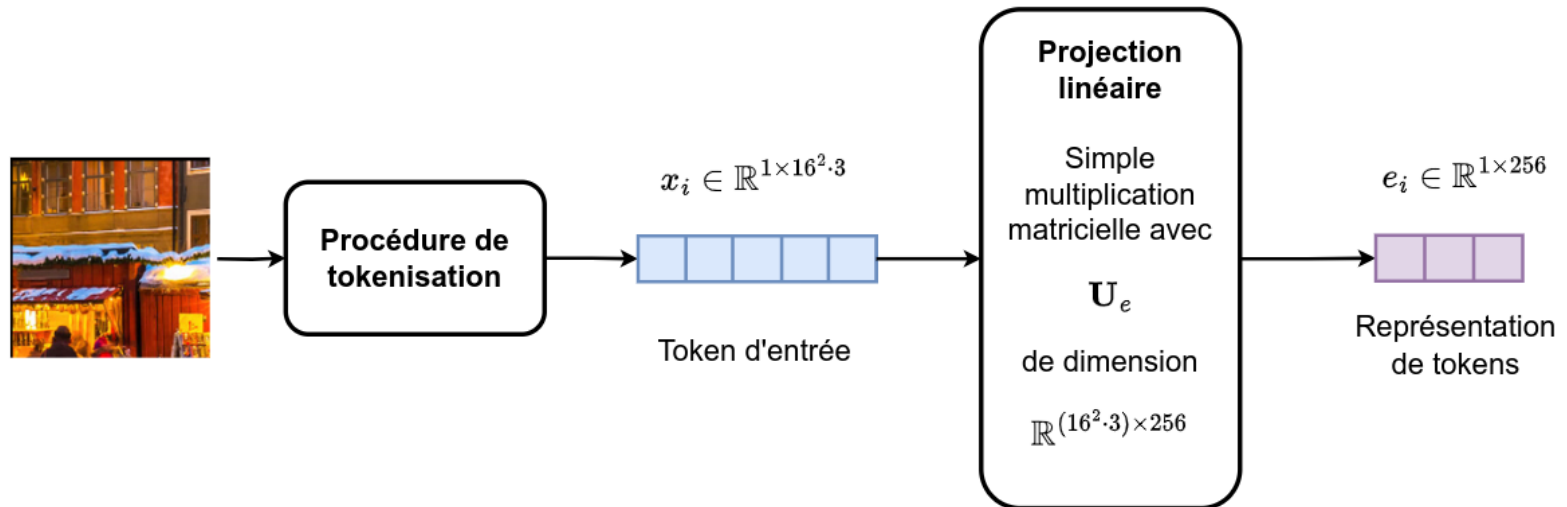
64 x 64 patches
de taille 16 x 16 px



► SAM: Segment Anything

→ Image encoder

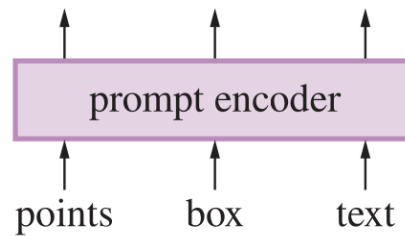
- Image d'entrée redimensionnées en 1024 x 1014 px
- Architecture: Vit-H/16 (token = patch de taille 16 x 16 px)
- Taille cachée D : 256



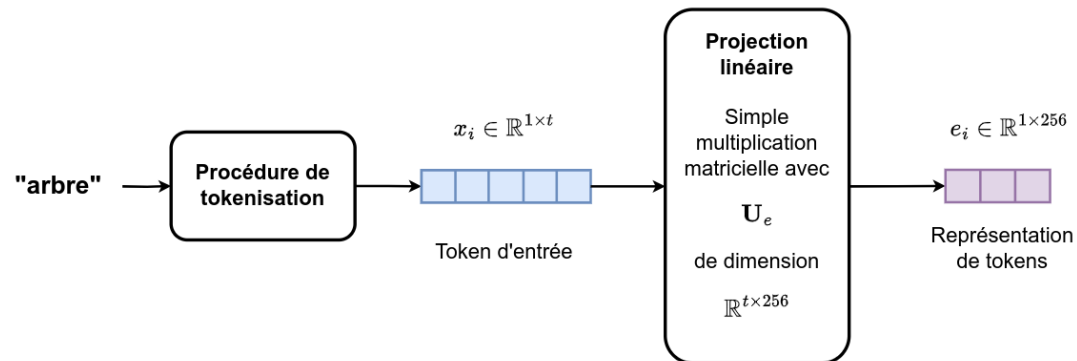
► SAM: Segment Anything

→ Prompt encoder

- Plusieurs prompts possibles: points, boîtes englobantes, texte
- Taille cachée D : 256



► Illustration pour le texte encoder

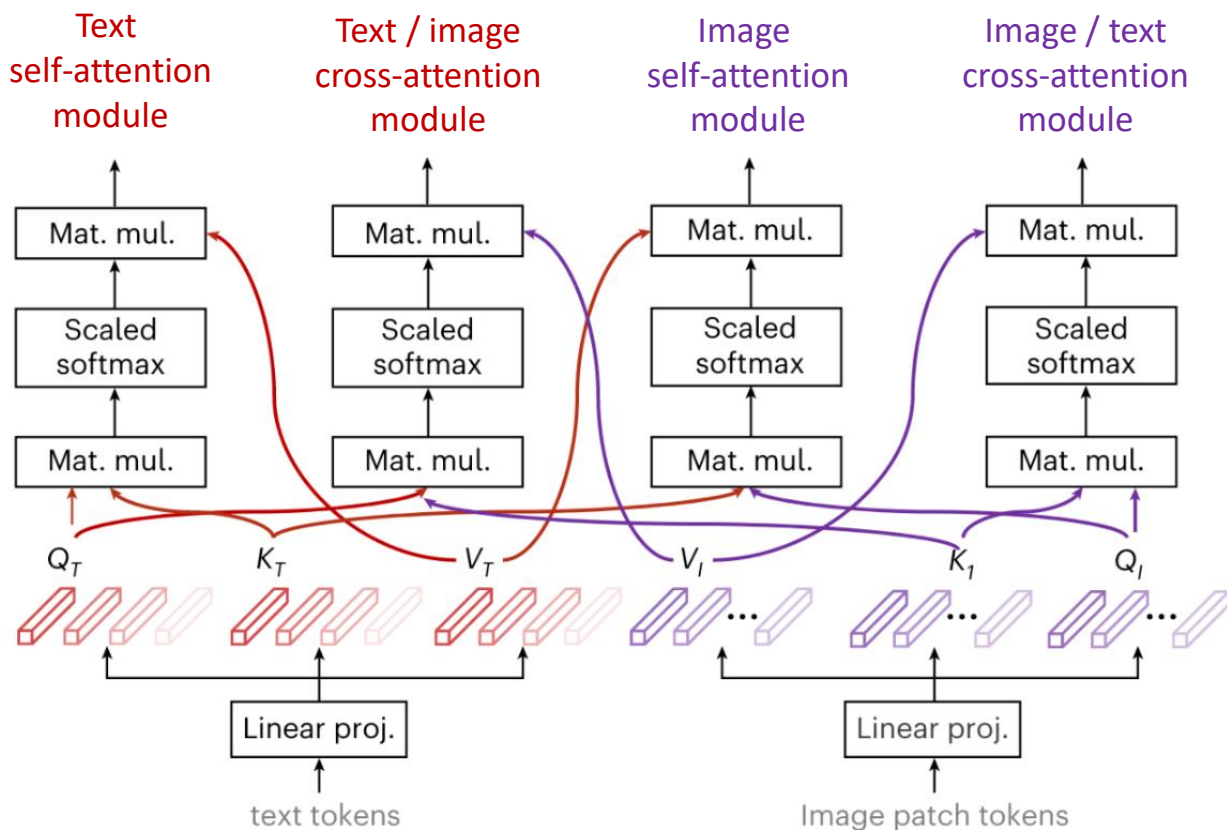


Transformer pour la segmentation

► SAM: Segment Anything

→ Mask decoder

► Utilise le principe d'attention croisée

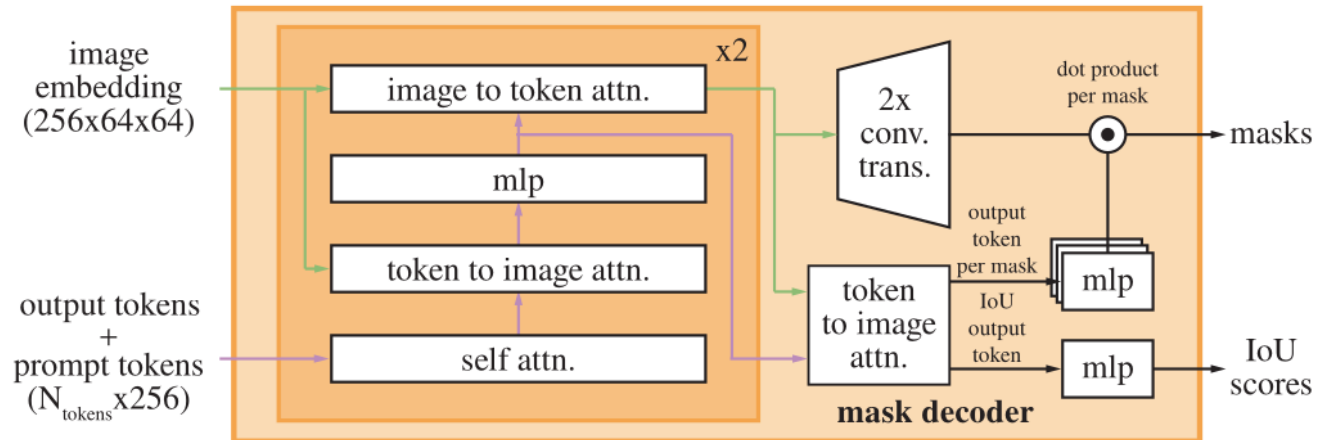


Transformer pour la segmentation

► SAM: Segment Anything

→ Mask decoder

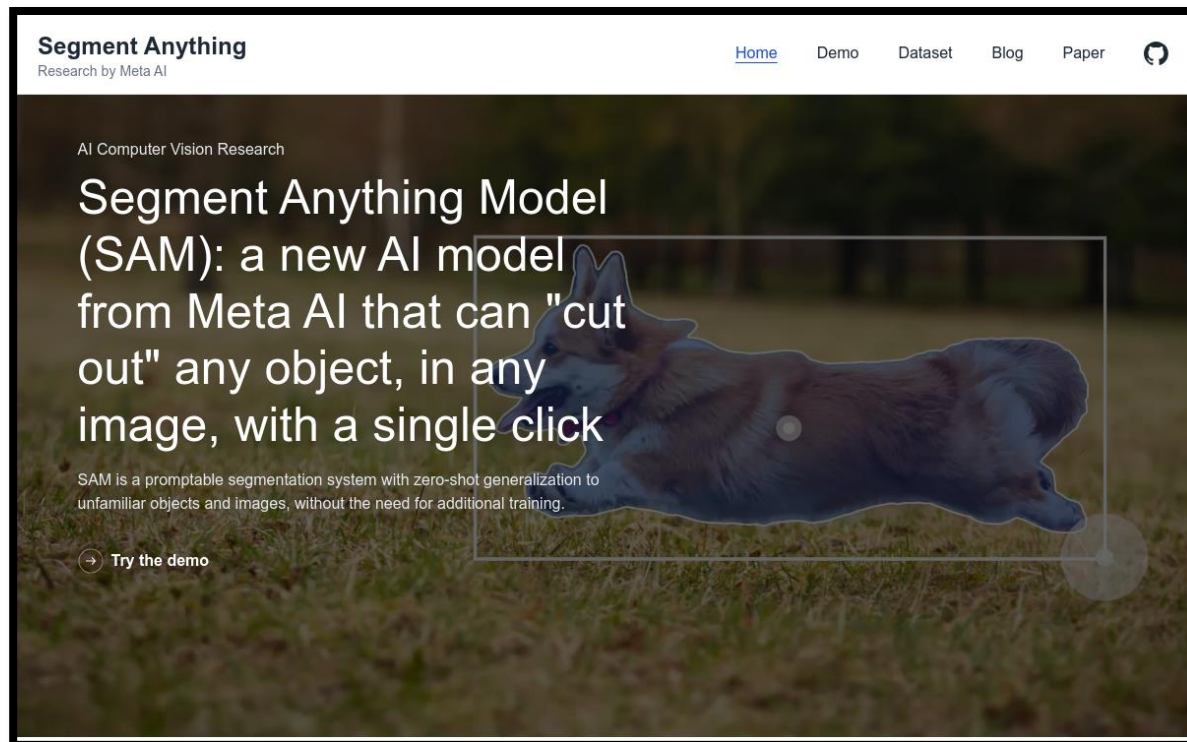
► Intégration d'un class token (output token)



► SAM: Segment Anything

→ Lien vers la démo

→ <https://segment-anything.com/>



That's all folks
