Modèle de diffusion

pour la génération d'image

Pierre-Antoine Jean
IMT Mines Alès

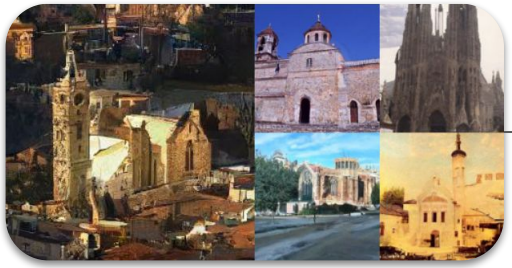
Sommaire

Introduction

Qu’est ce qu’un modèle de diffusion ?

- Un “modèle de diffusion” n’est pas un modèle à part entière mais une **manière d’entraîner** des modèles de *Deep Learning*
- Les modèles issus de cet entraînement sont dit **génératifs** et sont capables de générer de nouvelles données qui ressemblent à celles du jeu de données original

Par exemple, un modèle génératif entraîné sur des images peut créer de nouvelles images réalistes qui n'existent pas dans le jeu de données d'origine



Échantillons d’église au sein du jeu de données *LSUN* utilisés pour entraîner *Stable Diffusion*

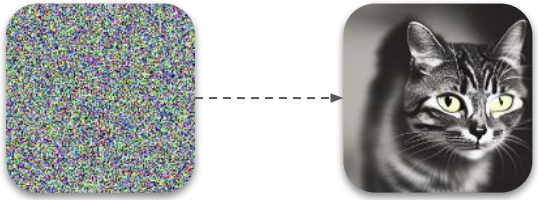
Génération d’une église via *Stable Diffusion*

Introduction

Comment fonctionne un modèle de diffusion pour la génération d'image ?

- Un modèle de diffusion, une fois entraîné, permet de générer des images à partir d'un **bruit gaussien pur** de même forme que l'image désirée

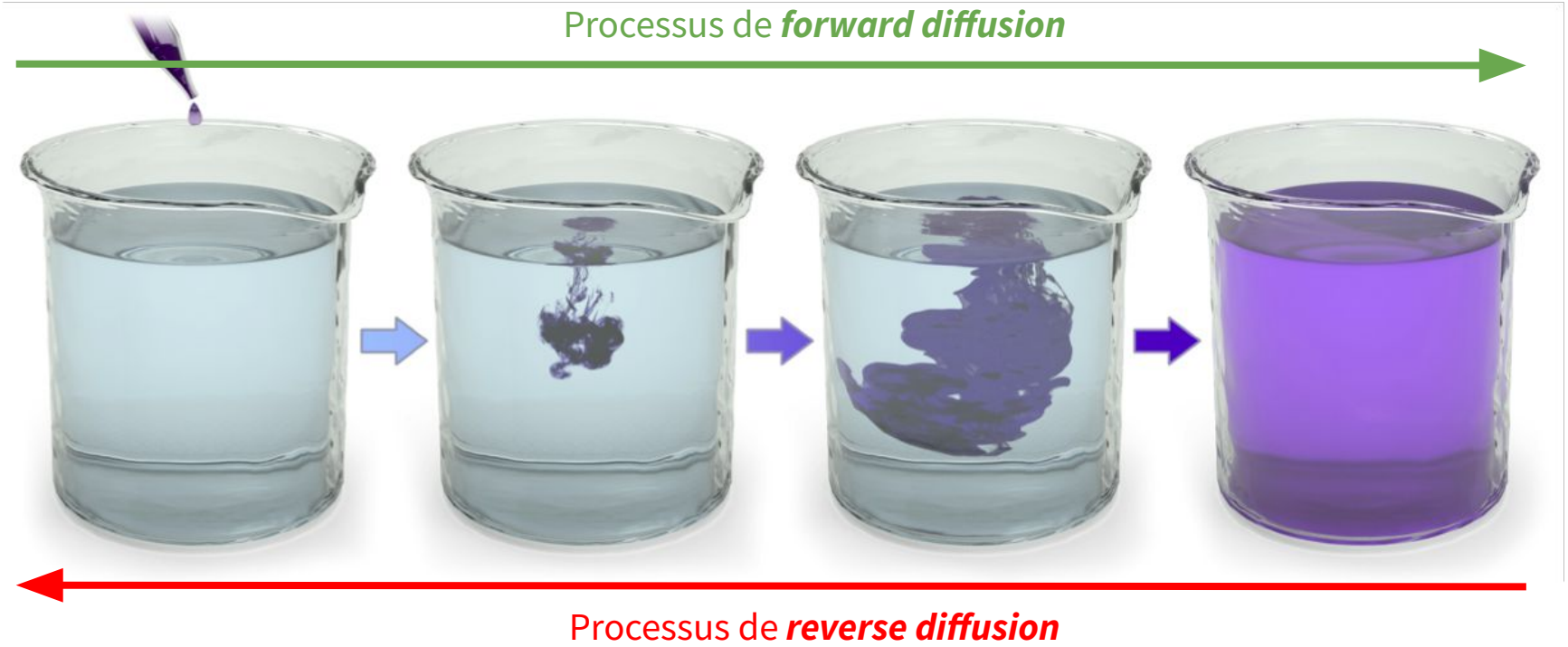
Les valeurs de chaque pixel sont tirées aléatoirement suivant une loi normale centrée réduite



- Un modèle de diffusion peut se décomposer en 2 phases principales :
  - L'entraînement, inclut le processus de ***forward diffusion*** qui consiste à ajouter du bruit à une image (en suivant un processus d'ajout successif) afin d'entraîner un modèle de *deep learning* à prédire le bruit total utilisé pour bruiter cette image
  - Le *sampling*, inclut le processus de ***reverse diffusion*** qui consiste à retirer une partie du bruit d'une image bruitée afin de générer une nouvelle image à partir d'un bruit aux dimensions de l'image désirée (en appliquant successivement ce processus)

Introduction

Intuition derrière les modèles de diffusion

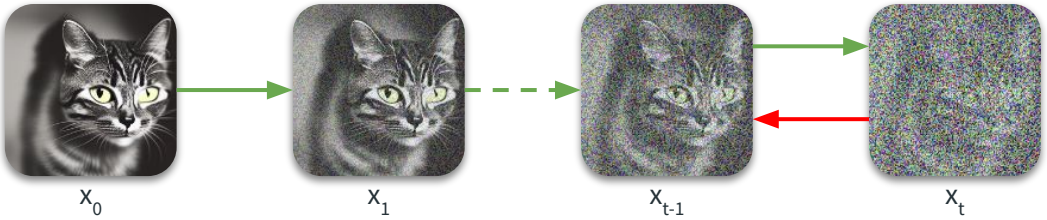


https://chemistryclinic.co.uk/1-3-diffusion/

Introduction

Vue d'ensemble des processus de *forward* et de *reverse diffusion*

- Le processus de ***forward diffusion***
  - Consiste à transformer une image nette de notre jeu de données en une image plus ou moins bruitée qui dépend d'un nombre d'itérations que l'on génère aléatoirement
  - Le bruit au sein de l'image x_t est la résultante d'une addition progressive d'une petite quantité de bruit à chaque itération depuis x_0



- Le processus de ***reverse diffusion***
  - Consiste à estimer une image moins bruitée x_{t-1} à partir d'une image plus bruitée x_t
  - Un modèle de *deep learning* est entraîné pour prédire le **bruit total** qui a été utilisé pour générer x_t
  - Une partie de ce bruit prédit, une fois soustrait à x_t, permet de retrouver x_{t-1}

Introduction

Algorithmes proposés par l'article "*Denoising Diffusion Probabilistic Models*" (DDPM)*

Apprendre à prédire le bruit utilisé pour générer une image bruitée

↻ Soustraire à une image bruitée une partie du bruit prédit

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Lien entre les algorithmes

Prédiction du bruit

Processus de ***forward diffusion***

Processus de ***reverse diffusion***

** Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. Advances in neural information processing systems, 33, 6840-6851.*

L’entraînement

Vue d’ensemble de l’algorithme

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on

$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$

6: **until** converged

L’entraînement

Tirage aléatoire d’une image x_0 du jeu de données

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

L’entraînement

Génération d’un nombre aléatoire entre 1 et *T* (hyperparamètre fixé à 1000*)

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t) \right\|^2$$
6: **until** converged

** Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. Advances in neural information processing systems, 33, 6840-6851.*

L’entraînement

Génération d’un bruit gaussien pur aux dimensions de x_0

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on

$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$

6: **until** converged

L’entraînement

Apprentissage d’un modèle *deep learning*

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on

$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$

6: **until** converged

L’entraînement

Apprentissage d’un modèle *deep learning*

$$\nabla_\theta \left\| \epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t\right)\right\|^2$$

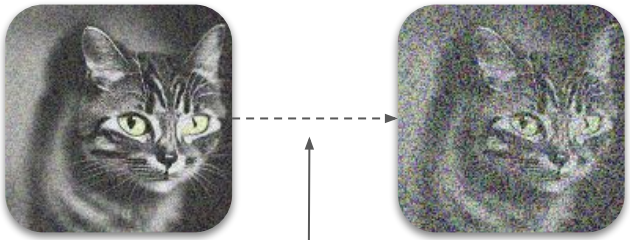
*Processus de **Forward diffusion** pour générer x_t*

Prédiction du bruit utilisé pour générer x_t

Mise à jour des poids du modèle en fonction du bruit généré et prédit

Le processus de *forward diffusion*

Principes



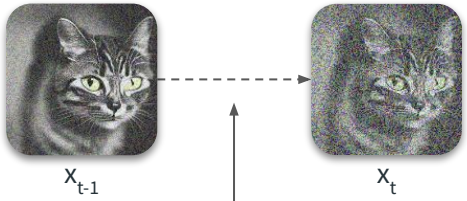
$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

distribution gaussienne *moyenne* *variance*

- Le processus de *forward diffusion* s'apparente à un processus markovien, où chaque étape x_t dépend uniquement de l'étape précédente x_{t-1}
- Ce processus suit une distribution normale dont la moyenne correspond aux valeurs des pixels de l'image précédente multipliées par un coefficient égal à la racine carré de $1 - \beta_t$ (avec β_t appelé *schedule*) et la variance est égale au coefficient β_t

Le processus de *forward diffusion*

Principes

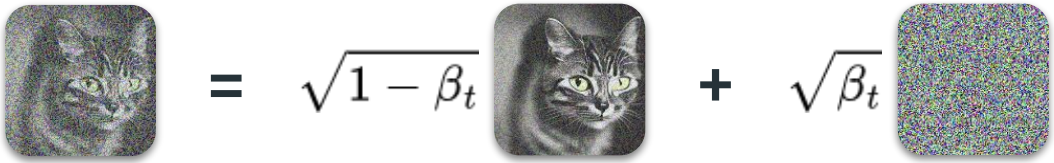


x_{t-1} x_t

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1}$$

avec $\beta_1 < \beta_2 < ... < \beta_T$ $\epsilon_{t-1} \sim \mathcal{N}(0, \mathcal{I})$

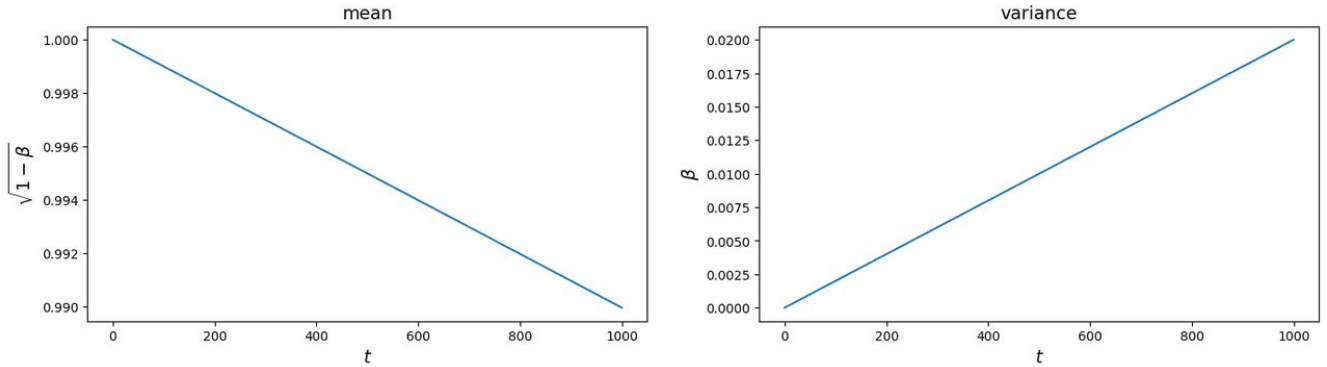


x_t = $\sqrt{1-\beta_t}$ x_{t-1} + $\sqrt{\beta_t}$ ϵ_{t-1} bruit gaussien

Le processus de *forward diffusion*

Coefficient βt (*schedule*)

- β_t est fonction de *t* et contrôle l'influence de l'image d'origine par rapport au bruit ajouté, il est compris entre 0 et 1 et augmente au fil des itérations
- *Ho et al.*[(1)] utilise un *linear schedule* (une fonction linéaire), avec β_t compris entre 0.0001 to 0.02



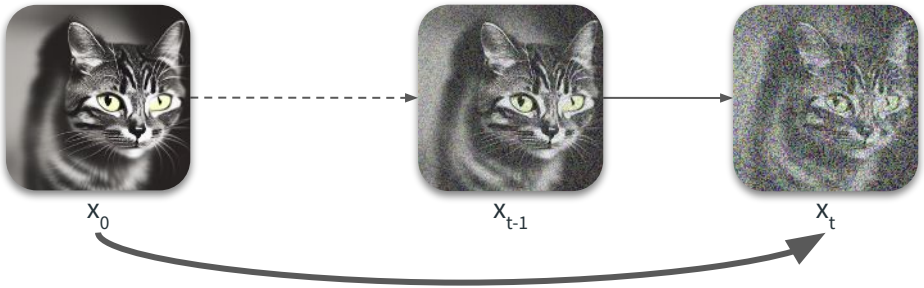
- A noter qu'en 2021, un papier d'OpenAI a proposé une autre manière de calculer le coefficient β_t permettant de réduire le nombre d'étapes total T[(2)]

[(1)] *Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. Advances in neural information processing systems, 33, 6840-6851.*
[(2)] *Nichol, A. Q., & Dhariwal, P. (2021, July). Improved denoising diffusion probabilistic models. In International conference on machine learning (pp. 8162-8171). PMLR.*

Le processus de *forward diffusion*

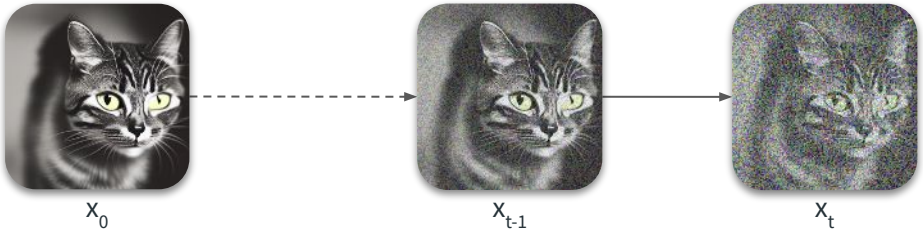
Généralisation de l'équation de *forward diffusion*

- Comme le processus de *forward diffusion* s'apparente à une chaîne de Markov, nous devons théoriquement réaliser *t* étapes d'ajout de bruit pour obtenir x_t depuis x_0
- Cependant, un modèle tel que Stable Diffusion est entraîné sur plusieurs centaines de millions d'images et il n'est pas envisageable de générer et stocker les *T* images bruitées de chacune des images du jeu de données (*e.g.* 1000 * 400M d'images)
- Pour répondre à cette problématique, les auteurs ont proposé une généralisation de l' équation du processus de *forward diffusion* pour générer l'image bruitée pour n'importe quel *t*, simplement à partir de l'image d'origine x_0



Le processus de *forward diffusion*

Généralisation de l'équation de *forward diffusion*



$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\overline{\alpha}_t}x_0, (1-\overline{\alpha}_t)I)$$

avec $\overline{\alpha}_t = \prod_{i=1}^{t}(1-\beta_i)$

Le processus de *forward diffusion*

Généralisation de l'équation de *forward diffusion*

Équation de base du processus de *forward diffusion* pour x_t

$$x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t} \epsilon_{t-1}$$

avec

$$\alpha_t = 1 - \beta_t$$

La même équation mais pour x_{t-1}

$$x_{t-1} = \sqrt{\alpha_{t-1}} x_{t-2} + \sqrt{1 - \alpha_{t-1}} \epsilon_{t-2}$$

Équation pour x_t en remplaçant x_{t-1} par son équation

$$x_t = \sqrt{\alpha_t \alpha_{t-1}} x_{t-2} + \sqrt{1 - \alpha_t \alpha_{t-1}} \overline{\epsilon}_{t-2}$$

avec $\overline{\epsilon}_{t-2}$ est la fusion de deux gaussiennes *

Répéter le processus précédent jusqu'à x_0

$$x_t = \sqrt{\overline{\alpha}_t} x_0 + \sqrt{1 - \overline{\alpha}_t} \epsilon$$

avec

$$\overline{\alpha}_t = \prod_{i=1}^{t} (1 - \beta_i)$$

$$q(x_t | x_0) = \mathcal{N}(x_t; \sqrt{\overline{\alpha}_t} x_0, (1 - \overline{\alpha}_t) I)$$

* Lorsqu'on fusionne deux gaussiennes $\mathcal{N}(0, \sigma_1^2 I)$ et $\mathcal{N}(0, \sigma_2^2 I)$ avec des variances différentes, la nouvelle distribution est égale à $\mathcal{N}(0, (\sigma_1^2 + \sigma_2^2) I)$. Dans ce cas, la déviation standard est égale à $\sqrt{(1 - \alpha_t) + \alpha_t (1 - \alpha_{t-1})} = \sqrt{1 - \alpha_t \alpha_{t-1}}$

Le processus de *forward diffusion*

Illustration de l'équation

$$x_t = \sqrt{\overline{\alpha}_t} x_0 + \sqrt{1 - \overline{\alpha}_t} \epsilon$$

x_t

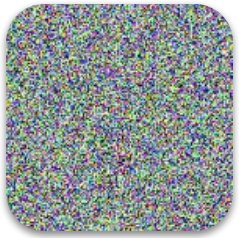
=

$\sqrt{\overline{\alpha}_t}$

x_0

+

$\sqrt{1 - \overline{\alpha}_t}$



ϵ, bruit gaussien

Illustration de l'algorithme d'entraînement

Pour rappel

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on

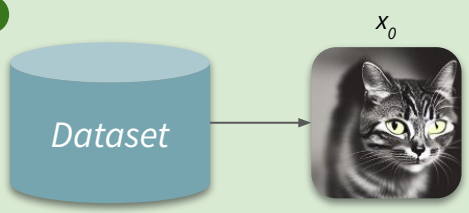
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$

6: **until** converged

Illustration de l'algorithme d'entraînement

Pour 1 répétition ! À multiplier par le nombre de *batches* et d'*epochs*

a



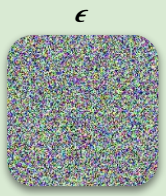
Récupération d'une image x_0

b

$t = 100$

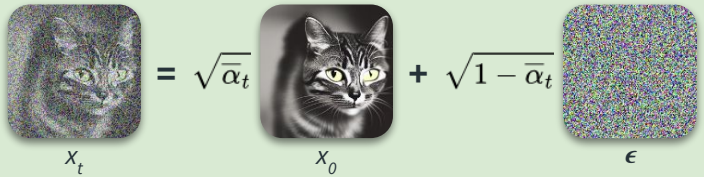
Génération d'un nombre aléatoire

c



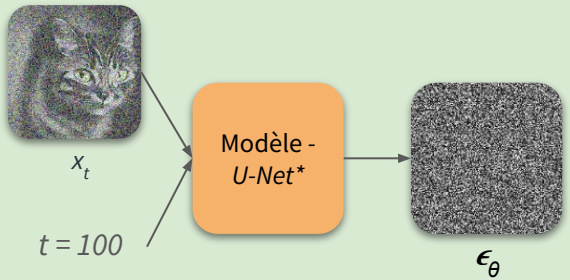
Génération d'un bruit gaussien

d



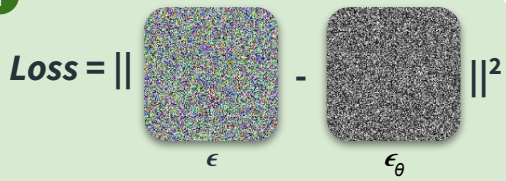
Calcul de x_t via le processus de *forward diffusion*

e



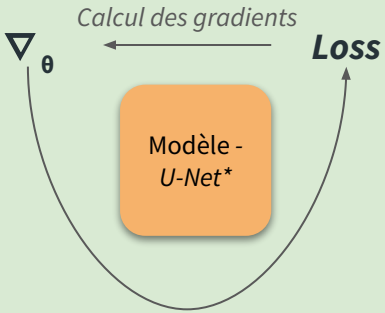
Prédiction du bruit ayant servi à la génération de x_t

f



Calcul de la fonction de perte (*MSE*)

g



Mise à jour des poids

* Le modèle le plus utilisé est un *U-Net* (un type de réseau de neurones convolutionnel avec une partie encodage et décodage)

Le *sampling*

Génération d'une image à partir d'un bruit gaussien pur

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \ldots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Le *sampling*

Génération d'un bruit aux dimensions de l'image désirée

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Le *sampling*

Génération d'un second bruit gaussien pour le calcul de x_{t-1}

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Le *sampling*

Calcul de x_{t-1} à l'aide du bruit prédit par le modèle *deep learning*

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Le *sampling*

Le processus de *reverse diffusion*

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$$

Prédiction du bruit

Une partie du bruit prédit est soustrait à x_t

Processus de reverse diffusion pour générer une image x_{t-1} un peu moins bruitée que l'image x_t

Le processus de *reverse diffusion*

Pourquoi avons nous besoin de prédire le bruit ?

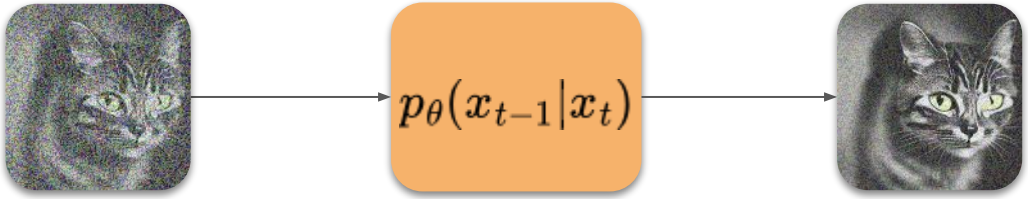
- L'idée du modèle de diffusion est d'inverser le processus de *forward diffusion* pour générer une véritable image à partir d'un bruit gaussien en entrée de la taille de l'image désirée
- Cependant, nous ne disposons pas d'une solution analytique pour résoudre $q(x_{t-1}|x_t)$, c'est pourquoi un modèle de *deep learning*, p_θ, est entraîné pour approximer cette fonction

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

Processus de *forward diffusion*

$$q(x_{t-1}|x_t) \approx p_\theta(x_{t-1}|x_t)$$

Approximation d'une fonction de débruitage



x_t

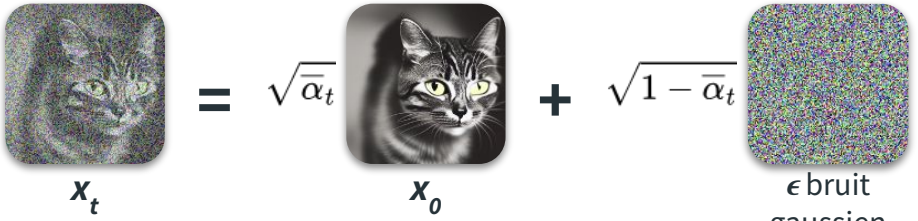
θ correspond aux poids du modèle appris pendant la phase d'entraînement

x_{t-1}

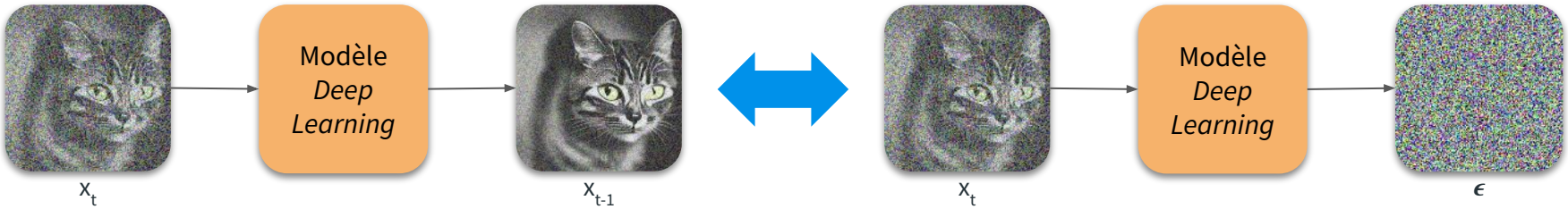
Le processus de *reverse diffusion*

Prédiction de x_{t-1} en prédisant ϵ

Rappel de l'équation du processus de *forward diffusion*



- Un modèle qui prend en entrée x_t et qui prédit la moyenne de la distribution de x_{t-1} est équivalent à un modèle qui prend également x_t et prédit ϵ, le bruit utilisé pour générer x_t



Le processus de *reverse diffusion*

Prédiction de x_{t-1} en prédisant ϵ

Approximation de la fonction *q*

$$q(x_{t-1}|x_t) \approx p_\theta(x_{t-1}|x_t)$$

En prédisant la moyenne d'une distribution normale

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

avec

$$\Sigma_\theta(x_t, t) = \sigma_t^2 I \begin{cases} \sigma_t^2 = \beta_t \\ \text{ou} \\ \sigma_t^2 = \widetilde{\beta}_t = \dfrac{1 - \overline{\alpha}_{t-1}}{1 - \overline{\alpha}_t}\beta_t \end{cases}$$

Le modèle de *deep learning* permet d'obtenir ϵ_θ

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1 - \overline{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$$

Pour obtenir x_{t-1}, on soustrait x_t avec une partie du bruit prédit

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1 - \overline{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sqrt{\beta_t}z$$

avec

$$\sigma_t^2 = \beta_t$$

Le processus de *reverse diffusion*

Illustration de l'équation

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}z$$

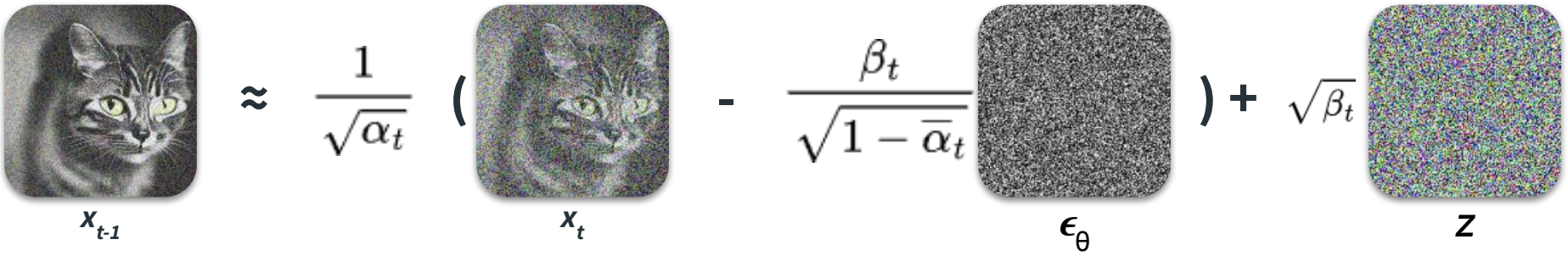
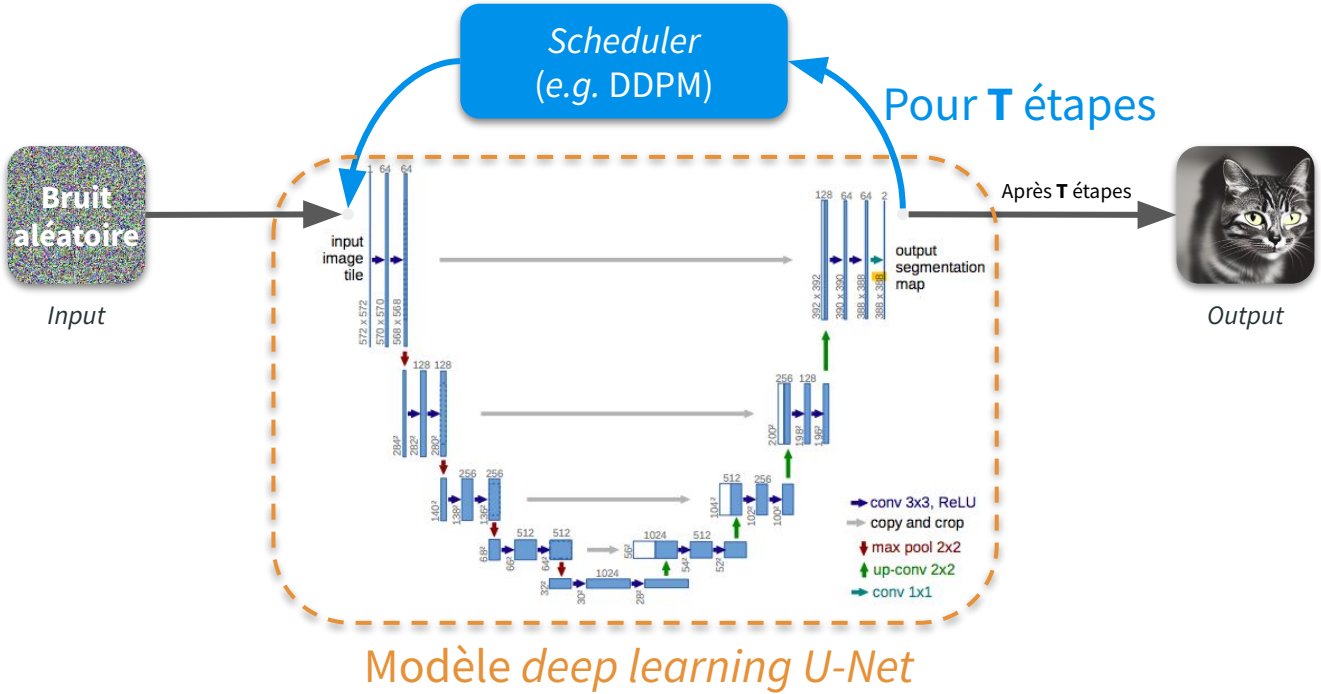


Illustration de l'algorithme de *sampling*

Génération d'une image à partir d'un bruit gaussien pur



Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18* (pp. 234-241). Springer International Publishing.

Stable diffusion

Les spécificités du modèle Stable diffusion

- Stable diffusion est un modèle génératif qui permet de créer des images, entre autres, à partir de descriptions textuelles
- Il est constitué de 3 composants essentiels :
  - Un composant pour prendre en charge le processus de *reverse diffusion*
  - Un composant d'encodage/décodage des images au sein d'un espace latent pour optimiser l'entraînement et l'inférence
  - Un composant pour conditionner le modèle afin d'intégrer les informations textuelles fournies, guidant ainsi la génération d'images en fonction du texte (un *text encoder*)

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 10684-10695).

Modèle de diffusion latent

Pourquoi travailler sur un espace latent

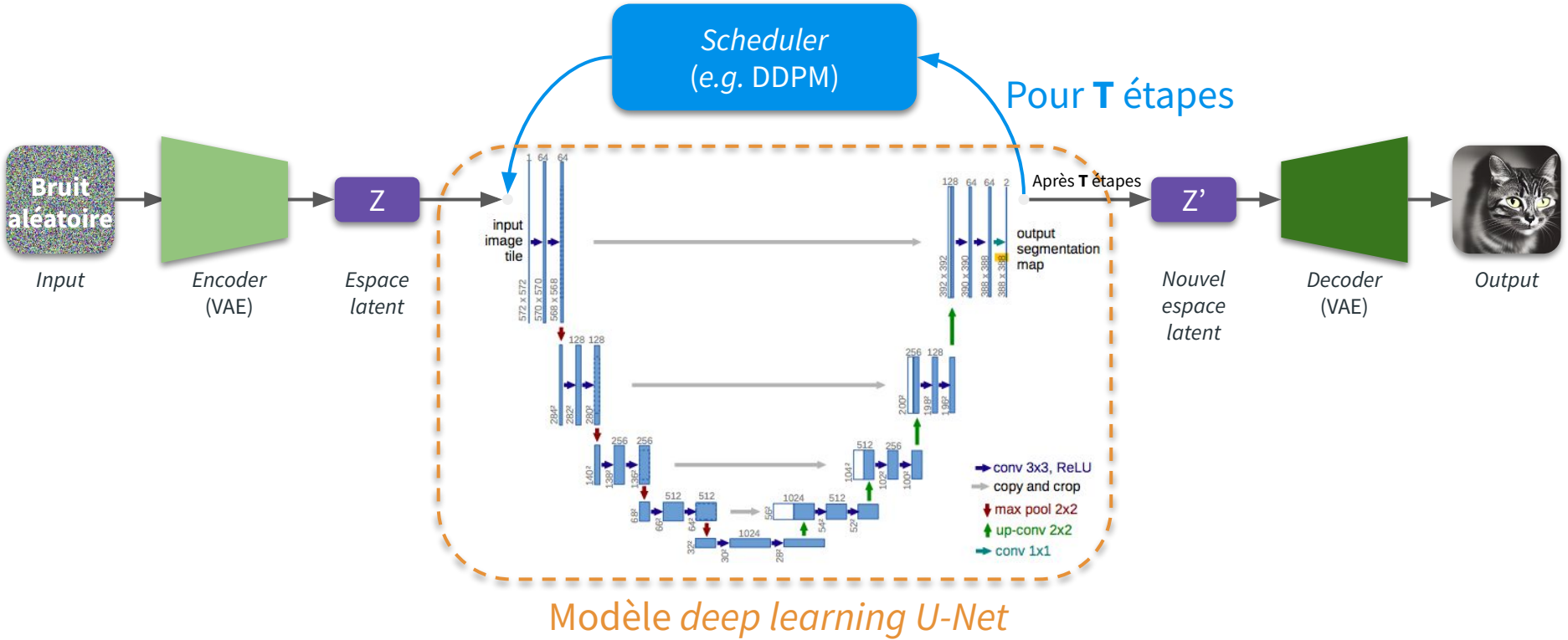
- Travailler directement dans l'espace des pixels est très coûteux en termes de temps et de ressources (l'espace des pixels d'une image carré de 512 pixels de côté avec 3 canaux RGB est équivalent à 512x512x3 = 786 432 dimensions)
- Pour accélérer l'entraînement et l'inférence, la notion d'espace latent est introduit au processus de diffusion

Dans le cas de la génération d'images, l'espace latent est une représentation compressée de l'image permettant ainsi de travailler avec beaucoup moins de données tout en conservant les informations visuelles essentielles

- *Stable Diffusion* est un modèle de diffusion latent et pour une image carré de 512 pixels de côté, il travaille sur un espace latent 48 fois plus petit (64x64x4 = 16384)
- C'est un **VAE** (*Variational autoencoder*) qui s'occupe de projeter les images dans un espace latent
- Il est entraîné séparément avant que le modèle de diffusion principal ne soit utilisé et durant cette phase d'entraînement, le VAE apprend à encoder et décoder des images de manière à minimiser la perte de reconstruction (c'est-à-dire la différence entre l'image originale et celle reconstruite)

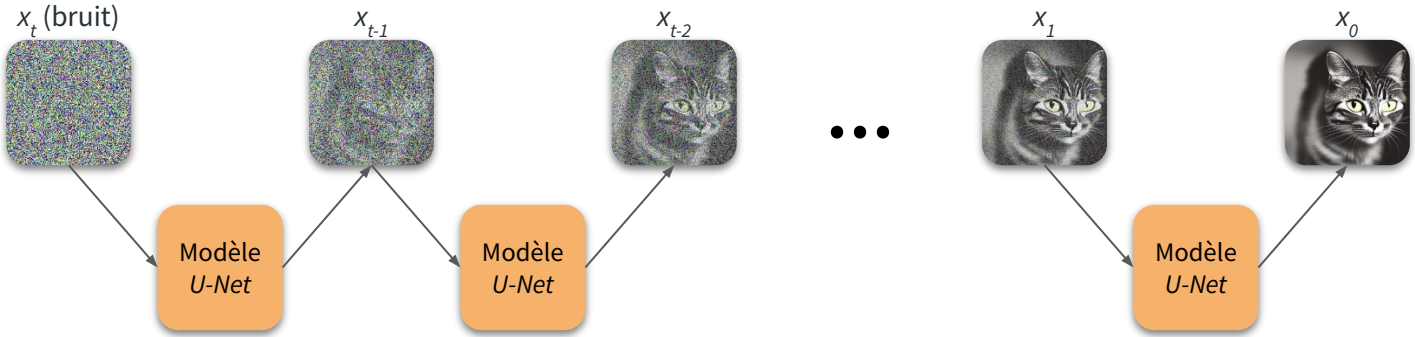
Illustration du modèle de diffusion latent

Vue d'ensemble



Modèle de diffusion latent conditionnel

Comment contrôler la sortie d'un modèle de diffusion ?



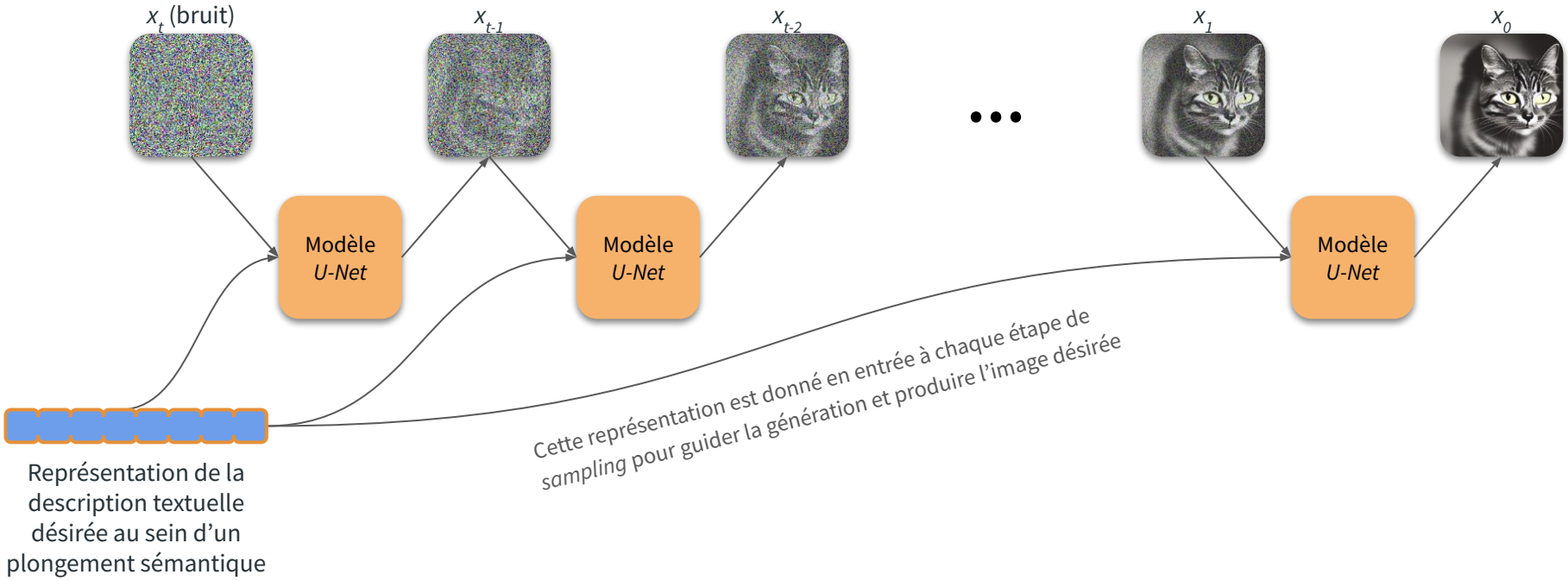
- Le processus de diffusion, en soi, ne permet pas de de contrôler directement l'image générée en sortie

  Par exemple, si le modèle est entraîné sur des images de chiens et de chats, il peut générer une image de l'un ou de l'autre sans que nous ayons de contrôle précis sur le résultat

- Même en utilisant le même bruit gaussien initial, le résultat final reste imprévisible, car à chaque étape du *sampling* (débruitage), un nouveau bruit gaussien est introduit
- L'objectif est donc de pouvoir **guider la diffusion** à l'aide de texte mais aussi à travers d'autres formes de conditionnement

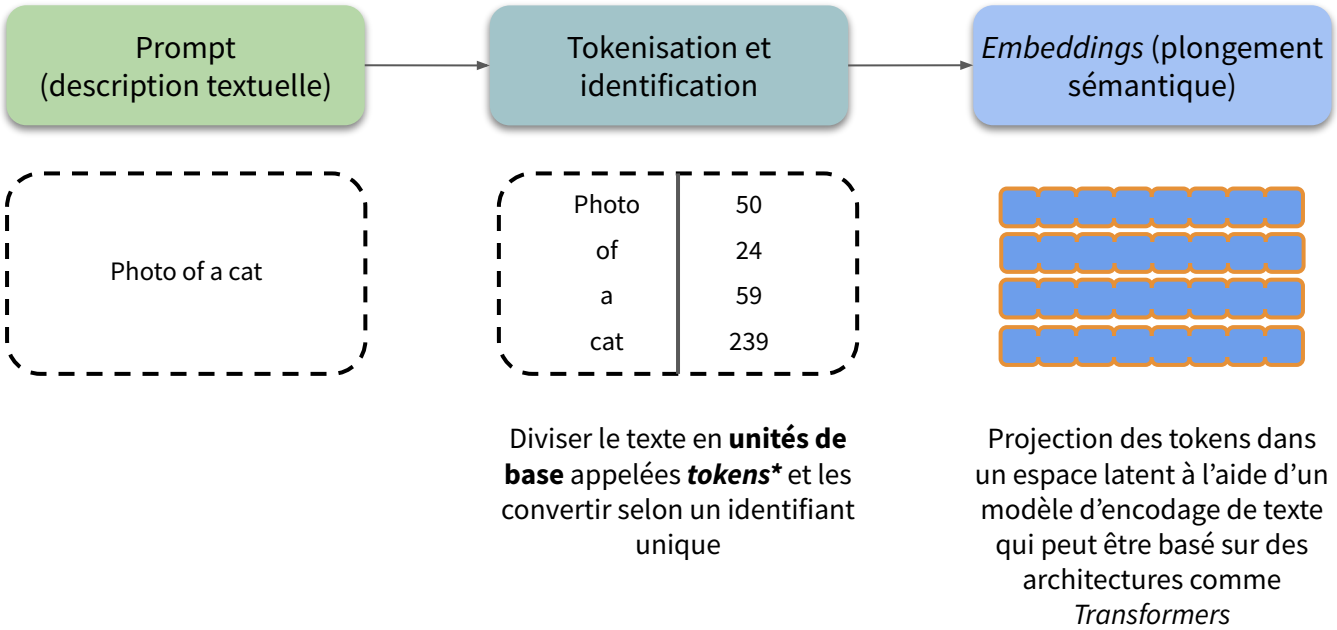
Modèle de diffusion latent conditionnel

Comment contrôler la sortie d'un modèle de diffusion à l'aide d'une description textuelle ?



Modèle de diffusion latent conditionnel

Du *pre-processing* à l'encodage du texte



Diviser le texte en **unités de base** appelées ***tokens**** et les convertir selon un identifiant unique

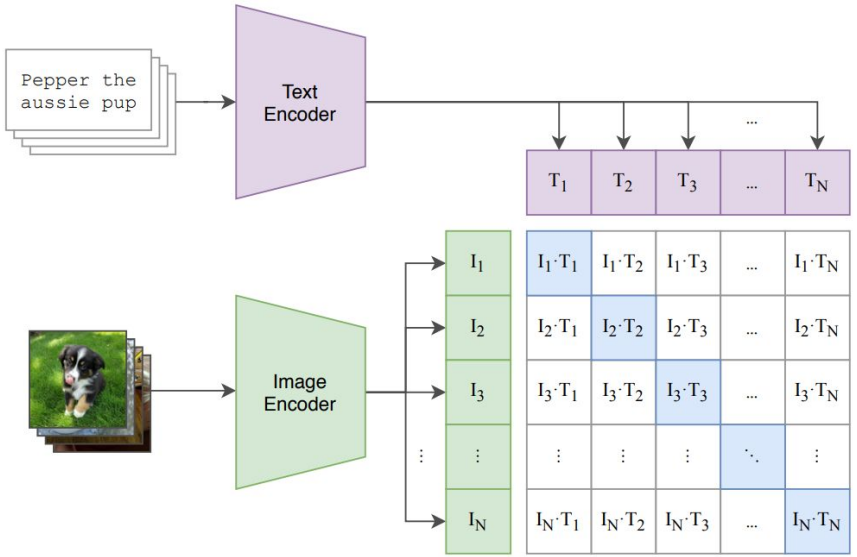
Projection des tokens dans un espace latent à l'aide d'un modèle d'encodage de texte qui peut être basé sur des architectures comme *Transformers*

* Un token n'est pas forcément un mot par exemple "skateboard" peut être représenté par 2 tokens "skate" et "board"

Modèle de diffusion latent conditionnel

Tokenisation et encodage du texte par le modèle CLIP

- C'est le modèle **CLIP** (*Contrastive Language-Image Pretraining*) et son tokenizer associé sont utilisés dans Stable Diffusion pour effectuer la tokenisation et le plongement sémantique du *pompt*
- CLIP, développé par OpenAI, est basé sur un modèle **Transformer** et son objectif principal est de créer un espace sémantique partagé dans lequel les textes et les images sont représentés par des vecteurs comparables
- La première version de Stable Diffusion utilise le modèle CLIP ViT-L/14 pour encoder les descriptions textuelles en vecteurs de dimension 768

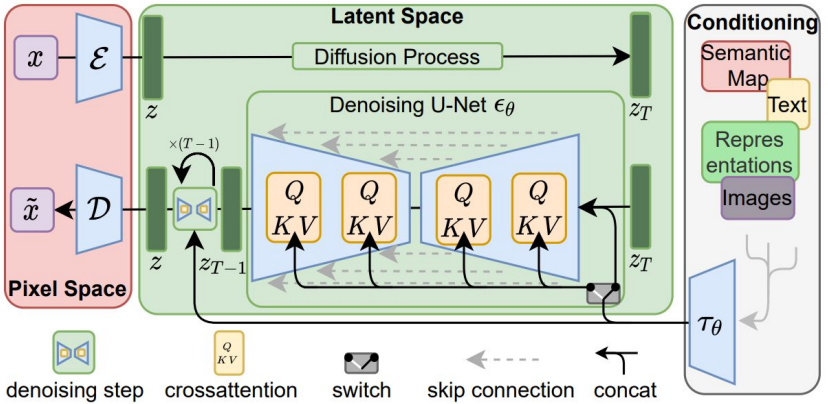


Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021, July). Learning transferable visual models from natural language supervision. In *International conference on machine learning* (pp. 8748-8763). PMLR.

Modèle de diffusion latent conditionnel

Conditionner l'étape de *sampling*

- Pendant le processus de débruitage, l'*embedding* du texte, qui contient les informations sémantiques du prompt, est injecté dans le modèle à travers une couche de *cross-attention*
- La ***cross-attention*** permet au modèle de "concentrer" son attention sur des parties spécifiques de l'embedding textuel pendant qu'il effectue le débruitage, en ajustant la génération d'image pour qu'elle corresponde mieux au contenu décrit par le prompt
- Cela permet de conditionner le débruitage en fonction de la description textuelle, afin que l'image générée soit alignée avec le sens du prompt



Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 10684-10695).

Mécanisme d’attention

Principes

- Le mécanisme d'attention est une technique utilisée dans les modèles de *deep learning*, notamment dans les **Transformers**, pour permettre au modèle de se concentrer sur des parties spécifiques des données en entrée

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

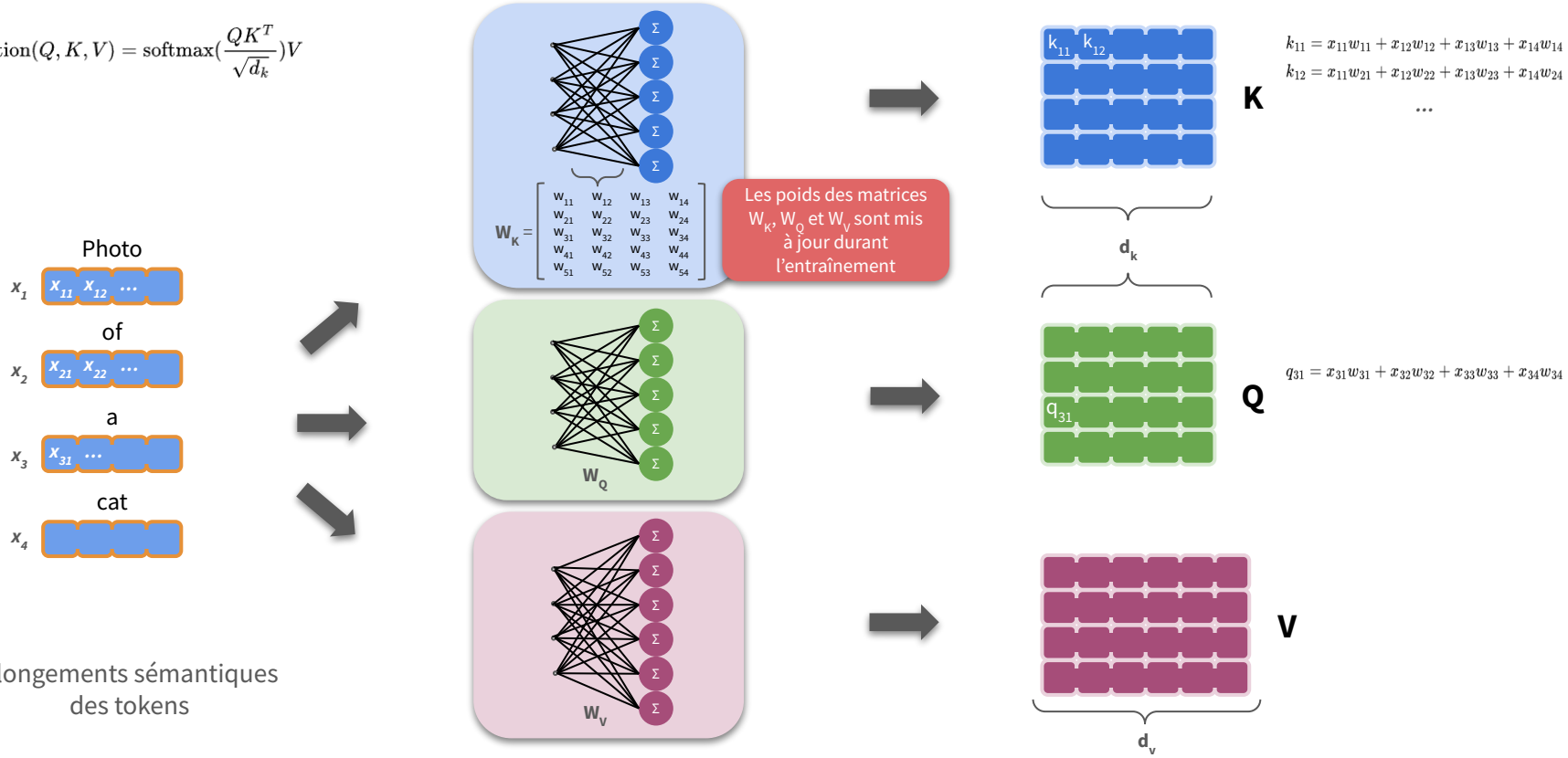
- Le mécanisme d’attention se décompose en 3 phases principales :
  - Calcul de 3 matrices (Q, K et V) : par la projection linéaire des vecteurs d’entrée (calcul de trois vecteurs pour chaque élément d'entrée)
  - Calcul des scores : pour chaque ligne de Q, on calcule un score en faisant un produit scalaire avec chaque élément de K pour déterminer l'importance relative de chaque élément d'entrée
  - Poids et Somme Pondérée : les scores sont ensuite normalisés (souvent avec une fonction softmax) pour obtenir des poids qui sont utilisés pour calculer une somme pondérée des éléments de V, mettant ainsi plus d'accent sur les éléments d'entrée jugés les plus pertinents.

Vaswani, A. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*.

Mécanisme d'attention

Chaque ligne des matrices K, Q et V est une transformation linéaire de chaque vecteur d'entrée

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



Mécanisme d'attention

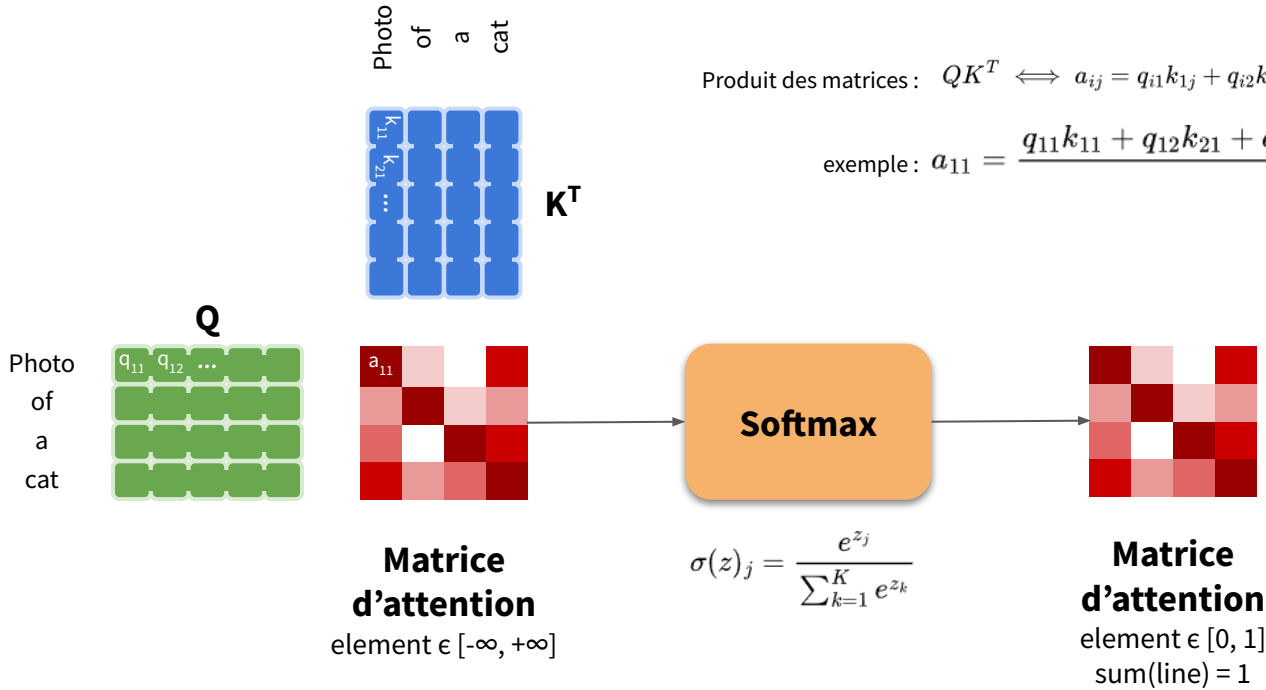
Chaque ligne des matrices K, Q et V est une transformation linéaire de chaque vecteur d'entrée

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

Produit des matrices : $QK^T \iff a_{ij} = q_{i1}k_{1j} + q_{i2}k_{2j} + \ldots + q_{ip}k_{pj} = \sum_{m=1}^{d_k} q_{im}k_{mj}$

exemple : $a_{11} = \frac{q_{11}k_{11} + q_{12}k_{21} + q_{13}k_{31} + q_{14}k_{41} + q_{15}k_{51}}{\sqrt{d_k}}$

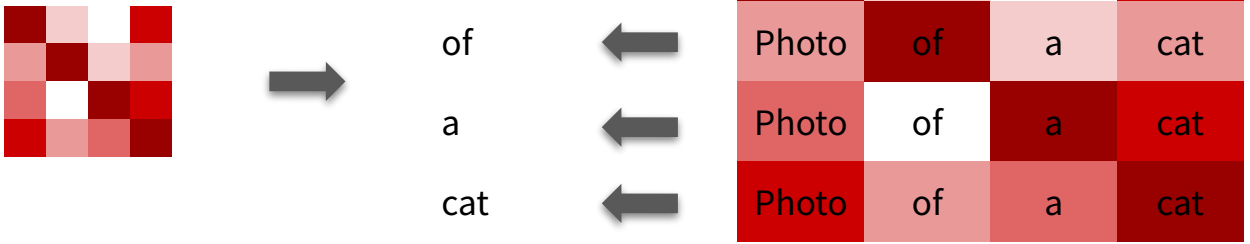
facteur de normalisation



Mécanisme d’attention

Intuition

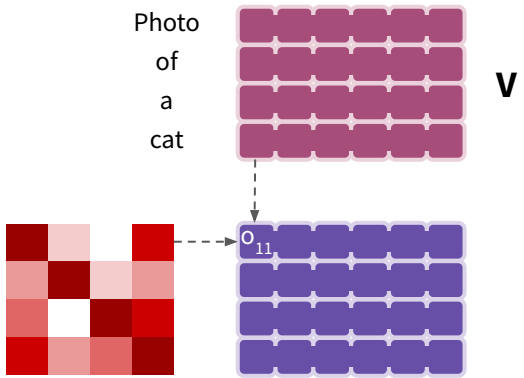
La matrice d'attention représente l'importance ou le poids que chaque token accorde aux autres tokens dans une séquence



Mécanisme d’attention

Chaque ligne des matrices K, Q et V est une transformation linéaire de chaque vecteur d’entrée

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



Matrice d’attention
element ϵ [0, 1]
sum(line) = 1

o_{11} =

X Photo

X of

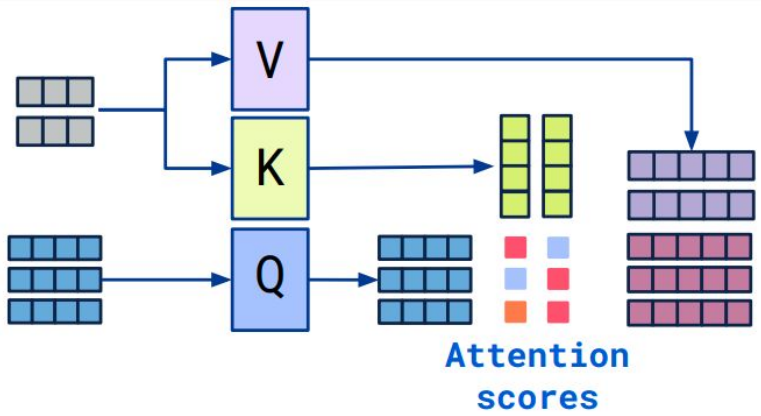
X a

X cat

Mécanisme de cross-attention

Guider le processus de débruitage à l'aide des plongements sémantiques du prompt

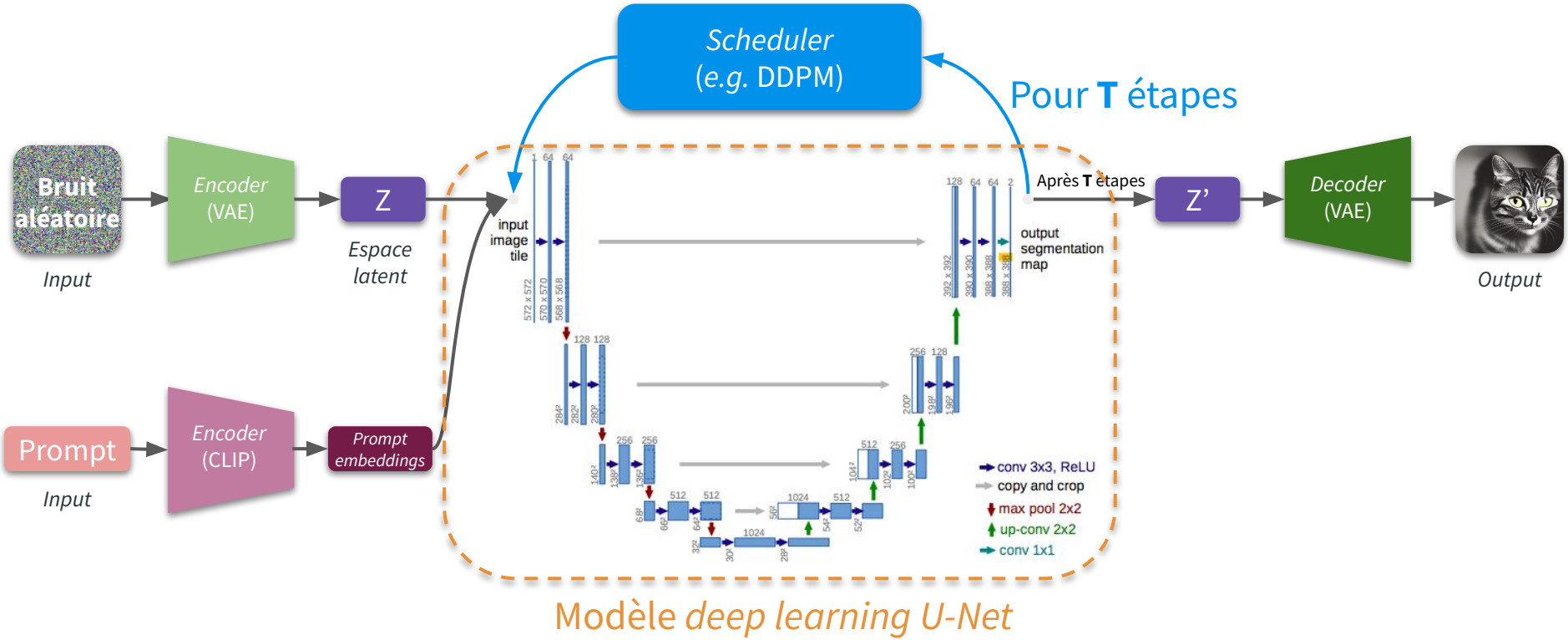
- Dans le mécanisme de cross-attention, qui est une variante du self-attention, les matrices Q, K, et V proviennent de deux ensembles différents de données
- Q provient de la représentation latente de l'image
- K et V sont dérivés des représentations textuelles encodées par un modèle de langage (CLIP)
- Les couches de cross-attention sont intégrées directement dans l'architecture du U-Net (à chaque niveau de résolution) pour permettre de guider la génération d'images à différents stades du processus de débruitage



https://vaclavkosar.com/ml/cross-attention-in-transformer-architecture

Illustration du modèle de diffusion latent conditionnel

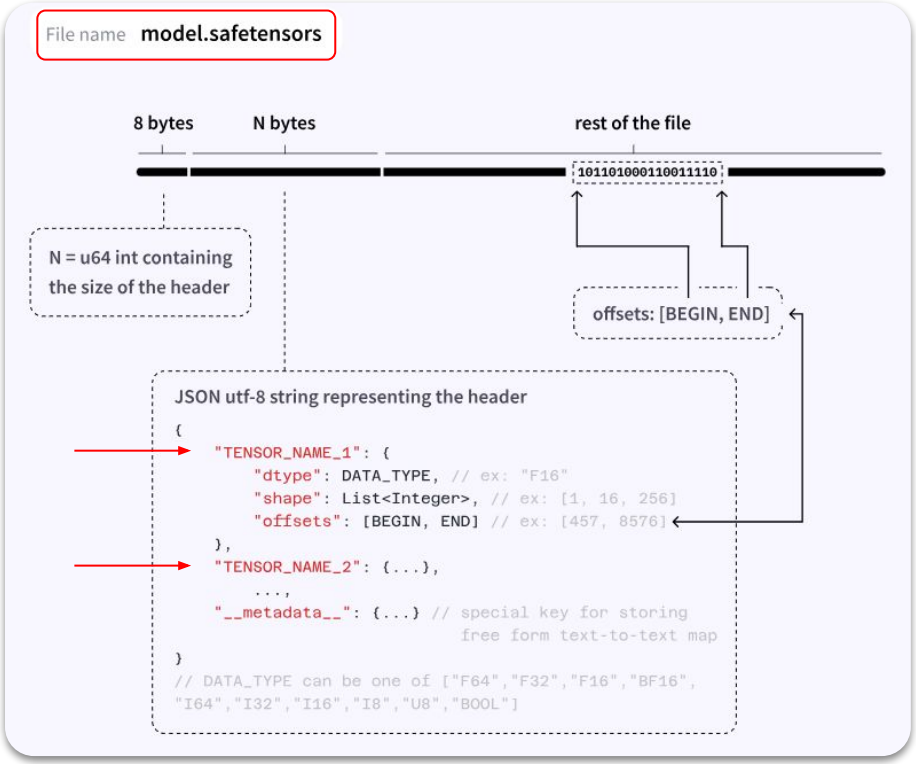
Vue d'ensemble



Checkpoint

Définition

- Un modèle *checkpoint* contient tous les paramètres appris durant l'entraînement et peut être utilisé pour l'inférence et le *fine-tuning*
- Le format courant d'un *checkpoint* est le *.safetensors* qui est un format rapide et simple pour sauvegarder et charger de manière sécurisée **un ou plusieurs tenseurs**
- Il est fréquent qu'un *checkpoint* inclut les poids des modèles U-Net, CLIP et VAE (fichier de plusieurs Gb)



Checkpoint et LoRA

Fine-tuning

- Le fine-tuning d'un modèle à partir d'un *checkpoint* implique la mise à jour de l'ensemble des poids du modèle (ceux du U-Net), ce qui entraîne un temps de calcul important et la génération d'un fichier de grande taille
- Les poids d'un modèle à affiner peuvent être représentés par l'addition de deux matrices, avec $W_0 \in R^{d \times k}$ représentant les poids pré-entraînés et ΔW la matrice mettant à jour ces poids

$$W' = W_0 + \Delta W$$

- La méthode LoRA (*Low-Rank Adaptation*), proposée dans un article de Microsoft*, introduit une technique d'entraînement légère pour effectuer le fine-tuning de grands modèles
- C'est une technique permettant de représenter la matrice de mise à jour des poids à l'aide d'une décomposition de bas rang

* Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2021). Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

LoRA

Équations et illustration

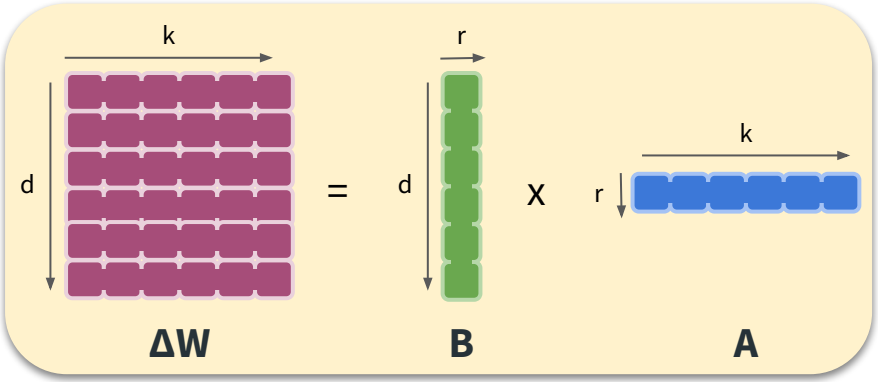
Mise à jour de la matrice de poids $W' = W_0 + \Delta W$

Décomposition matricielle de ΔW $\Delta W = B \times A$

avec $B \in \mathbb{R}^{d \times r}$

$A \in \mathbb{R}^{r \times k}$

$r \ll \min(k, d)$



- En fixant W_0, il est possible d'affiner les matrices A et B, qui contiennent les paramètres entraînables pour l'adaptation du modèle initial

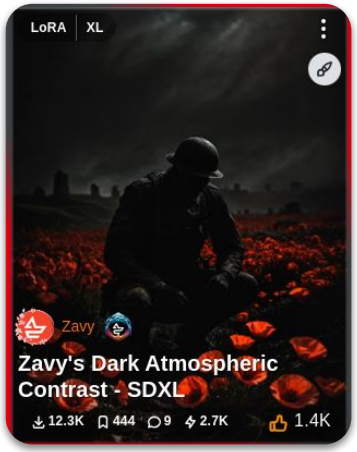
Phase de *feed forward* avec les matrices $h = W'x = W_0x + BAx$

https://machinelearningmastery.com/using-lora-in-stable-diffusion/

LoRA

Customiser son style

- Au sein de Stable Diffusion, la technique LoRA s'applique aux matrices W_Q, W_K et W_V de la couche de cross-attention
- Par conséquent, ces matrices décomposés A et B permettent d'obtenir des fichiers plus légers et plus rapidement
- Les LoRA sont généralement employés pour ajouter un style particulier à la génération



Quelques liens supplémentaires

- https://lilianweng.github.io/posts/2021-07-11-diffusion-models/
- https://medium.com/@kemalpiro/step-by-step-visual-introduction-to-diffusion-models-235942d2f15c
- https://www.youtube.com/watch?v=V8L766qXmUA
- https://www.youtube.com/watch?v=EvH-qJKSZV8&list=PLlI0-qAzf2SZQ_ZRAh4u4zbb-MQHOW7IZ
- https://www.youtube.com/watch?v=ZBKpAp_6TGI
- https://medium.com/@lovelyndavid/self-attention-a-step-by-step-guide-to-calculating-the-context-vector-3d4622600aac
- https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html#:~:text=Defining%20the%20Weight%20Matrices&text=Self%2Dattention%20utilizes%20three%20weight,components%20of%20the%20sequence%2C%20respectively.