RAPPORT DE STAGE

MÉTHODES FRUGALES POUR L'INPAINTING

MATHIS WAUQUIEZ

Vers de l'inpainting vidéo efficace

Mai 2025 – Septembre 2025

SUPERVISEURS:
Yann Gousseau
Alasdair Newson
Andrés Almansa

ÉTABLISSEMENT:
Télécom Paris- LTCI - Équipe IMAGES

DATE DE DÉBUT DE STAGE:
Mai 2025

Mathis Wauquiez: *Méthodes frugales pour l'inpainting*,
Vers de l'inpainting vidéo efficace
Ponts-MVA

"Inpainting, the technique of modifying an image in an undetectable form, is as ancient as art itself." [6]

REMERCIEMENTS

Je souhaite exprimer ma profonde gratitude à mes encadrants, avec qui j'ai passé cinq mois particulièrement enrichissants. Je remercie tout d'abord Yann Gousseau[1], qui m'a accueilli au sein du LTCI, pour son accompagnement constant et ses conseils avisés. Merci également à Alasdair Newson[2] pour sa disponibilité, son investissement et la confiance qu'il m'a accordée. J'adresse aussi mes remerciements à Andrés Almansa[3], pour la richesse de ses propositions et pour les nombreuses références qui ont nourri ma curiosité intellectuelle. J'aimerais également remercier Nicolas Cherel. Nicolas est un ancien doctorant de mes superviseurs, et ce stage est la continuation de ses travaux de thèse, nommée "Internal methods for the generation and inpainting of images and videos" [16].

Enfin, merci à toute l'équipe du LTCI pour son accueil et sa bienveillance, ainsi qu'à toutes les personnes avec lesquelles j'ai pu échanger.

1 LTCI, Télécom Paris
2 ISIR, Sorbonne Université
3 MAP5, Université Paris Descartes & CNRS

TABLE DES MATIÈRES

Partie I

INTRODUCTION

INTRODUCTION

Effacer un objet indésirable d'une image prend aujourd'hui quelques secondes sur un smartphone. Faire de même sur une vidéo peut nécessiter plusieurs dizaines de minutes de calcul sur un GPU récent. Ce fossé d'accessibilité n'est pas un hasard : l'inpainting vidéo repose majoritairement sur des modèles de diffusion dont le coût computationnel croît linéairement avec la dimension temporelle. Bien que visuellement impressionnants, ces modèles restent hors de portée des ordinateurs grand public et posent des questions d'impact énergétique et donc environnemental non négligeables. La démocratisation de l'édition vidéo exige donc de repenser l'efficacité de l'inférence, sans sacrifier la qualité visuelle. L'inpainting complétion de régions masquées dans un média guidée potentiellement par du texte, une image de référence ou d'autres modalités – dépasse largement ce cas d'usage. Cette formulation générique permet de résoudre diverses tâches : suppression d'objets, super-résolution, restauration de zones endommagées, ou encore retouche artistique. Portée par un fort intérêt du grand public et par des applications en imagerie médicale [87], télédétection et apprentissage de représentations, la recherche sur l'inpainting s'est fortement accrue ces dernières années.

Toutefois, les architectures développées deviennent toujours plus gourmandes, notamment à cause de la popularisation des méthodes par diffusion au détriment de méthodes moins coûteuses comme les approches par patchs et par équations différentielles et des lois d'échelle des réseaux de neurones. Ainsi, des solutions récentes et populaires comme Repaint [64] reposent principalement sur l'utilisation de larges modèles de diffusion, aux coûts énergétiques et environnementaux relativement élevés et non adaptés aux ordinateurs grand public, en particulier pour l'inpainting vidéo.
Face à ce constat, nous posons la question suivante :

Comment développer des modèles frugaux pour l'inpainting de vidéos ?

Des modèles frugaux permettraient d'accroître l'efficacité énergétique, de réduire les coûts d'inférence et de permettre le développement de solutions commerciales fonctionnant sur ordinateurs grand public.

Pour répondre à cette question, nous identifions deux axes complémentaires, nous concentrant sur les approches par modèles de diffusion. Premièrement, réduire le nombre d'évaluations de fonctions (NFE, *Number of Function Evaluations*) nécessaires aux modèles de diffusion. Deuxièmement, concevoir une architecture de débruitage vidéo économe en ressources de calcul. Ces deux directions structurent nos contributions.
Afin de réduire le nombre d'évaluations de fonctions, nous avons adopté le *Flow Matching*, un cadre de modélisation générative récent étroitement lié aux modèles de diffusion. Ce cadre a récemment gagné en popularité grâce à plusieurs atouts: sa simplicité mathématique, sa capacité à traiter des distributions source non-gaussiennes, ses liens avec le transport optimal dynamique ainsi que son échantillonnage rapide. De plus, ce cadre est plutôt populaire pour les méthodes de distillation, ouvrant la voie vers des travaux futurs pour d'avantage diminuer le nombre d'évaluations de fonctions.

Contribution logicielle. Afin de pouvoir réaliser rapidement un grand nombre d'expériences, nous avons développé une bibliothèque complète et modulaire pour l'entraînement de modèles de *Flow Matching*. Contrairement aux implémentations existantes souvent limitées à des cas d'usage spécifiques, notre bibliothèque offre une flexibilité remarquable en intégrant l'ensemble des paramétrisations de l'état de l'art. Il est ainsi possible de configurer librement le *scheduler*, les distributions source et cible, la stratégie d'échantillonnage temporel, la fonction de perte, la paramétrisation du modèle, le solveur d'équations différentielles, et les modes de guidage (avec ou sans classifieur). Cette bibliothèque a permis de réaliser nos expériences de manière systématique et reproductible, tout en facilitant l'exploration de nouvelles configurations. Elle est disponible sur ce dépôt: https://github.com/mathis-wauquiez/FlowMatchingLibrary.

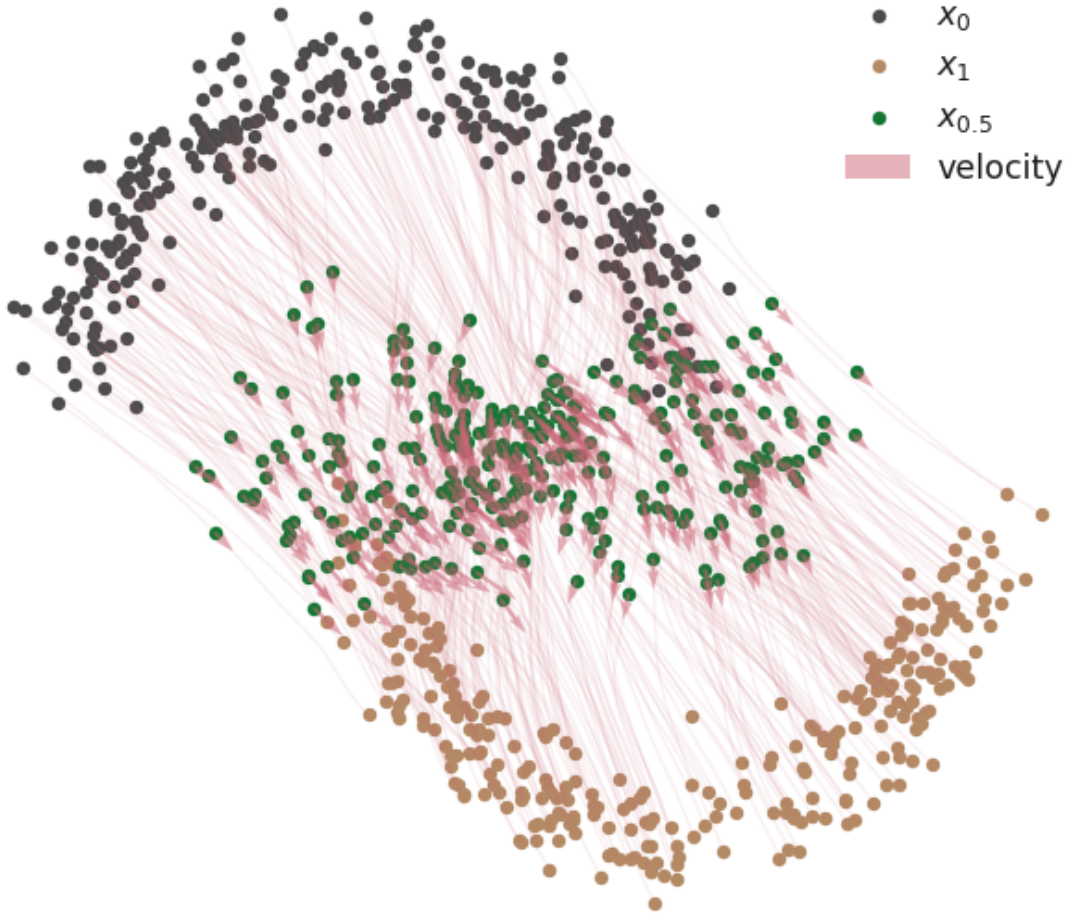
Inpainting d'images et transport optimal. Une seconde partie de ce stage et de ce rapport est le développement de modèles d'inpainting d'images rapide, basé sur la théorie du *Flow Matching*. Dans ce cadre, nous avons dressé l'état de l'art ainsi qu'une partie de l'histoire de ce domaine, avant de proposer deux méthodologies différentes d'inpainting d'image uniquement basées sur l'auto-similarité des images, n'incorporant pas d'a priori venant de larges bases de données pouvant parfois poser des problèmes éthiques ou légaux.

Inpainting de vidéos Nous généralisons ensuite la méthode la plus fructueuse à la vidéo, et prouvons ainsi qu'il est possible d'entraîner des modèles pour faire de l'inpainting vidéo en utilisant moins de $500,000$ paramètres, en atteignant une bonne fidélité visuelle.

Vers une nouvelle attention linéaire Enfin, nous proposons une méthode d'attention efficace, à la complexité linéaire, particulièrement adaptée pour l'inpainting d'images et de vidéos. Nous évaluerons celle-ci en utilisant le cadre d'inpainting développé dans les parties précédentes. Une attention particulière a été portée à l'efficacité ainsi qu'à la capacité de passage à l'échelle de cette couche, implémentée en CUDA.

Partie II

FLOW MATCHING



Dans les deux prochains chapitres, nous allons formuler la tâche d'inpainting comme une tâche d'échantillonnage conditionnel: nous allons échantillonner des images $x \in \mathbb{R}^d$ sachant le masque m et la condition (extérieur du masque, ainsi qu'un autre potentiel signal) $y \in \mathbb{R}^l$ depuis la distribution $\mathbb{P}[x \mid M = m, Y = y]$. Afin de pouvoir échantillonner depuis cette distribution de probabilités, nous allons devoir nous familiariser avec le cadre de Flow Matching (**FM**).
Le Flow Matching [1, 2, 53, 54] est un cadre simple et élégant introduit en 2022 ayant repoussé l'état de l'art dans de nombreuses applications, dont la génération d'images [26], de vidéos [89, 121] et d'audio [102]. Il est basé sur l'apprentissage d'un champ de vitesses u_t définissant un flot ψ_t, solution d'une équation différentielle ordinaire $\frac{d}{dt}\psi_t = u_t(\psi_t)$, analogue à l'équation pouvant régir le mouvement de particules soumises à un champ de vitesses $u_t(x)$. Le flot est une transformation continue et bijective transformant un échantillon $X_0 \sim p$ en un échantillon $X_1 = \psi_1(X_0)$ tel que X_1 soit distribué selon une distribution désirée q. Un exemple classique serait d'avoir $p(.) = \mathcal{N}(. \mid 0, I_d)$ et q la distribution des images naturelles, nous permettant ainsi d'échantillonner des images naturelles à souhait. Les modèles de flots furent d'abord introduits en 2018 sous la forme des *Continuous Normalizing Flows* [15, 31], qui sont une extension continue directe des *Normalizing Flows* [47, 79]. Dans leur première forme, ils étaient entraînés par maximisation de la vraisemblance $p(X_1)$. Ce processus était très inefficace car il nécessitait de rétropropager les gradients à travers chaque évaluation de fonction de la simulation de l'équation différentielle. Des travaux subséquents ont cherché à supprimer ce besoin, en changeant l'objectif à apprendre [4, 83], ce qui a évolué en la formulation moderne du **FM**[1, 2, 53, 54, 59, 77, 98], qui, comme les modèles de diffusion [35, 90, 91, 93], ne nécessite plus de simulation durant l'entraînement et est donc bien plus pratique. Bon nombre de généralisations de ce cadre existent, notamment à des ponts de Schrödinger [1, 8, 9, 55], des états discrets (en NLP par exemple) [13, 28], ou même des processus de markov à temps continu généraux [37], mais ne seront pas traitées ici.
Chronologiquement, les modèles de diffusion furent le premier cadre de processus de Markov en temps continu à pouvoir être entraîné sans simulation. Dans leur première formulation, ils étaient entraînés comme des processus de Markov discrets [35, 90], avant d'être formulés en temps continu [93] grâce à une formulation sous la forme d'équations différentielles stochastiques de la forme $dx = f(x,t)dt + g(t)dW_t$, avec:

- $f(x,t)$ le drift
- $g(t)$ le coefficient de bruit
- W_t un Brownien
- $(f(x,t), g(t))$ le scheduler

Cette formulation est extrêmement similaire avec celle des modèles de flot, de la forme $\frac{d}{dt}\psi_t = u_t(\psi_t)$, dans laquelle nous retrouvons la même notion de drift, u_t, mais sans le mouvement Brownien. Cependant, une différence conceptuelle notable existe: le cadre de Flow Matching construit d'abord une interpolation entre les distributions avant d'en extraire son champ de vitesses par différentiation (on part d'une solution pour en construire l'ODE), là où les modèles de diffusion partent d'une équation de diffusion pour ensuite déterminer une forme close de l'expression de la distribution de probabilités de x_t sachant l'échantillon cible x_1, $\mathbb{P}[x_t \mid x_1] = \mathcal{N}(x_t; \alpha_t x_1, \sigma_t I_d)$ (on établit une équation différentielle puis en étudie la solution). Formellement, cette dernière expression est une interpolation entre une distribution gaussienne d'origine et la distribution cible: $X_t = \alpha_t X_1 + \sigma_t X_0$ avec $X_0 \sim \mathcal{N}(.; 0, I_d)$, échantillonnée durant l'entraînement en conditionnant sur $X_1 = x_1$.

.1 PRINCIPALES NOTATIONS

Ce chapitre introduit le cadre du *Flow Matching*. Pour faciliter la lecture, nous listons ici les notations essentielles, dans leur ordre d'apparition approximatif :

Distributions et échantillonnage

p, q	Distributions source (facile à échantillonner) et cible (à apprendre)
X_0, X_1, X_t	Variables aléatoires aux temps $t \in [0,1]$; x_0, x_1, x_t leurs réalisations
$\Pi_{0,1}(x_0, x_1)$	Couplage des distributions source et cible (souvent indépendant : $p \times q$)
$p_t(x)$	**Chemin de probabilités** : interpolation entre p (en $t = 0$) et q (en $t = 1$)
$p_{t\|Z}(x\|z)$	Chemin conditionnel sachant $Z = z$

Flot et vitesses (concepts centraux)

ψ_t	**Flot** : transformation continue de x_0 vers x_1, solution d'une ODE
$u_t(x)$	**Champ de vitesses** théorique en (x, t), à apprendre
$u_t^\theta(x)$	Approximation neuronale de u_t, paramétrée par θ
$\psi_t(x_0, x_1)$	**Interpolant** : chemin reliant x_0 à x_1 (ex: $(1-t)x_0 + tx_1$)

Scheduler d'interpolation linéaire

α_t, β_t	Coefficients du scheduler : $X_t = \alpha_t X_0 + \beta_t X_1$
	Conditions aux bords : $(\alpha_0, \beta_0) = (1, 0)$ et $(\alpha_1, \beta_1) = (0, 1)$
$\dot{\alpha}_t, \dot{\beta}_t$	Dérivées temporelles (apparaissent dans u_t)

Paramétrisations alternatives du champ

$x_{0\|t}(x), x_{1\|t}(x)$	Prédiction de $\mathbb{E}[X_0\|X_t = x]$ (bruit) ou $\mathbb{E}[X_1\|X_t = x]$ (signal)
$s_t(x)$	Score : $s_t(x) = \nabla_x \log p_t(x)$

Conditionnement et guidance

Z	Variable de conditionnement (souvent $Z = X_1$ ou $Z = (X_0, X_1)$)
Y, y	Observation externe (masque, classe, texte) pour guidance
λ	Intensité de la *classifier-free guidance*

Apprentissage

$\mathcal{L}_{\text{CFM}}(\theta)$	Fonction de perte de *Conditional Flow Matching*
ρ	Distribution d'échantillonnage du temps t (uniforme ou logit-normale)

Conventions : Les majuscules (X) dénotent des variables aléatoires, les minuscules (x) leurs réalisations. L'exposant θ indique un modèle appris. Le symbole $f_\# \mu$ (push-forward) désigne la loi de $f(X)$ quand $X \sim \mu$.

.2 CADRE THÉORIQUE

Dans cette section, nous allons présenter le cadre théorique du *Flow Matching*. Nous l'avons également illustré, sous la forme d'une vidéo de cinq minutes, disponible ici, créée grâce à la librairie implémentée.

.2.1 *Présentation du problème*

Nous avons à notre disposition des réalisations d'une variable aléatoire $(X_0, X_1) \sim \Pi_{0,1}(X_0, X_1)$ constituant notre jeu de données. Souvent, ce couplage de données est indépendant: on notera alors $\Pi_{0,1}(X_0, X_1) = p(X_0) \cdot q(X_1)$, avec $X_0 \sim p$, $X_1 \sim q$. L'exemple typique est quand p est une gaussienne isotrope, facile à échantillonner, et q la distribution des images naturelles, mais l'on pourrait également avoir q la distribution des images en couleur et p la distribution des images en noir et blanc, nous permettant ainsi

d'échantillonner des images en couleur conditionnellement à leur version en noir et blanc. Nous allons alors constuire les objets suivants:

- Une interpolation $p_t = \mathcal{L}(X_t)$ entre la distribution d'origine p et la distribution cible q.
- Un champ de vitesses théorique u_t, dont le flot associé ψ_t, solution de l'équation différentielle $\frac{d}{dt}\psi_t = u_t(\psi_t)$ vérifie $\psi_t(X_0) \overset{\mathcal{L}}{=} X_t$.
- Une approximation u_t^θ de u_t, entraînée en minimisant une fonction de perte.

Nous échantillonnerons alors depuis q en échantillonnant $X_0 \sim p$ et intégrant l'équation différentielle $\frac{d}{dt}\psi_t(X_0) = u_t^\theta(\psi_t(X_0))$.

.2.2 *Construction du chemin de probabilités*

Nous appellons chemin de probabilités les distributions $(p_t)_{0\leqslant t\leqslant 1}$ du processus de Markov en temps continu $(X_t)_{0\leqslant t\leqslant 1}$. Similairement à ce qui se fait traditionnellement dans les modèles de diffusion, les modèles de flot construisent cette distribution de façon conditionnelle. En tant que premier exemple, nous allons imaginer conditionner par rapport à $X_1 = x_1$, comme dans les modèles de diffusion, bien que d'autres conditionnements $Z = z$ sont possibles. Cela nous donnera le chemin de probabilités marginal $p_t(x)$ par intégration du chemin de probabilités conditionnel $p_{t|1}(x|x_1)$:

$$p_t(x) = \int p_{t|1}(x \mid x_1) q(x_1)\, dx_1$$

En diffusion, on aurait typiquement $p_{t|1}(x|x_1) = \mathcal{N}(x; \alpha_t x_1, \sigma_t I_d)$.
Pour résoudre le problème de *Flow Matching*, on voudrait que p_t satisfasse:

$$p_0 = p, \quad p_1 = q,$$

ie. nous voulons que p_t corresponde à une interpolation entre la distribution source p à $t = 0$ vers la distribution cible q à $t = 1$. Ces conditions se rapportent au chemin de probabilités marginal: nous cherchons à satisfaire

$$p_{0|1}(x|x_1) = \Pi_{0|1}(x|x_1), \quad p_{1|1}(x|x_1) = \delta_{x_1}(x),$$

avec $\Pi_{0|1}(x|x_1) = \frac{\Pi_{0,1}(x,x_1)}{q(x_1)}$.
De nombreuses interpolations sont possibles, et seront discutées dans la section .2.6.
Un exemple de chemin de probabilités conditionnel particulièrement populaire est $p_t(.) = \mathcal{N}\left(. \mid tx_1, (1-t)^2 I_d\right)$, introduit par *Rectified Flow* [59], qui correspond à une interpolation linéaire $X_t = (1-t)X_0 + tX_1$ entre une loi normale standard et la distribution cible.
De façon générale, nous pouvons conditionner notre chemin de probabilités p_t par une variable aléatoire Z. Des choix typiques de conditionnement sont $Z = X_1$ [35, 53, 91, 92] ou $Z = (X_1, X_0)$ [2, 59].

.2.3 *Dérivation du champ de vitesse*

La dérivation d'un champ de vitesses u_t générant notre interpolation p_t se décompose en trois étapes principales :

1. **Choix du conditionnement Z :** sélection d'une variable auxiliaire pour créer l'interpolation conditionnelle, possédant une forme close.
2. **Construction du champ conditionnel :** détermination d'un champ de vitesses conditionnel $u_t(x_t \mid Z)$ qui génère le chemin de probabilité conditionnel $p_t(x_t \mid Z)$.

3. **Marginalisation :** intégration du champ conditionnel sur la distribution de Z pour obtenir le champ marginal.

Nous précisons d'abord le sens du terme « générer » dans ce contexte :

Définition .2.1 (Champ générateur). Un champ de vitesses u_t *génère* la famille de distributions p_t si la solution ψ_t du système différentiel

$$\begin{cases} \frac{d}{dt}\psi_t(x) = u_t(\psi_t(x)), \\ \psi_0(x) = x, \end{cases} \tag{1}$$

vérifie $(\psi_t)_{\#}p_0 = p_t$ pour tout $t \in [0,1]$.

Définition .2.2 (Champ générateur conditionnel). De manière similaire, un champ de vitesses conditionnel $u_t(\cdot \mid z)$ génère la famille conditionnelle $p_{t|Z}(\cdot \mid z)$ si la solution $\psi_{t|z}$ du système

$$\begin{cases} \frac{d}{dt}\psi_{t|z}(x) = u_t(\psi_{t|z}(x) \mid z), \\ \psi_{0|z}(x) = x, \end{cases} \tag{2}$$

vérifie $(\psi_{t|z})_{\#}p_{0|z} = p_{t|z}$ pour tout $t \in [0,1]$ et z dans le support de Z.

Le résultat principal établit la relation entre champ conditionnel et champ marginal :

Théorème .2.1 (Marginalisation du champ de vitesse). Sous les hypothèses de régularité suivantes :

(i) $p_{t|Z}(x \mid z) \in \mathcal{C}^1([0,1) \times \mathbb{R}^d)$ pour tout z,

(ii) $u_t(x \mid z) \in \mathcal{C}^1([0,1) \times \mathbb{R}^d)$ pour tout z,

(iii) p_Z admet un support compact,

(iv) $p_t(x) > 0$ pour tout $(x,t) \in \mathbb{R}^d \times [0,1)$,

(v) $u_t(x \mid z)$ génère $p_{t|Z}(\cdot \mid z)$ au sens de la Définition .2.2,

le champ de vitesses marginal défini par

$$\boxed{u_t(x) = \mathbb{E}[u_t(X_t \mid Z) \mid X_t = x]} \tag{3}$$

génère le champ de probabilités marginal p_t au sens de la Définition .2.1.

Preuve .2.1. La démonstration complète est donnée dans la section 4.4 de [54], ainsi que dans la vidéo explicative, vers 2 minutes 50. L'idée clé repose sur l'équation de continuité et le théorème de conservation de la masse [101].

Remarque. L'équation (3) exprime que le champ marginal est l'espérance conditionnelle du champ conditionnel sachant la position $X_t = x$. Cette formulation sera cruciale pour l'apprentissage du champ u_t^θ.

.2.4 *Apprentissage du champ de vitesses*

Maintenant que nous avons déterminé une vélocité cible u_t générant le chemin de probabilité p_t désiré, nous allons nous intéresser à comment, en pratique, apprendre un modèle u_t^θ le plus proche possible de u_t. Ce problème est non trivial, étant donné que l'on ne peut pas explicitement calculer u_t - l'intégration sur Z étant insoluble en grandes dimensions.
Heureusement, nous savons que:

$$u_t(x) = \mathbb{E}_Z[u_t(X_t \mid Z) \mid X_t = x] \tag{4}$$
$$= \arg\min_{v \in \mathbb{R}^d} \mathbb{E}_Z\left[\|u_t(X_t \mid Z) - v\|^2 \,\middle|\, X_t = x\right] \tag{5}$$
$$= \arg\min_{v:\, \mathbb{R}^d \to \mathbb{R}^d} \mathbb{E}_Z\left[\|u_t(X_t \mid Z) - v(X_t)\|^2\right] \tag{6}$$
$$\tag{7}$$

Donc, nous pouvons apprendre un modèle u_t^θ minimisant $\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,Z,X_t\sim p_{t|Z}(.|Z)}\left[\|u_t(X_t|Z) - u_t^\theta(X_t)\|^2\right]$.

OBJECTIF D'APPRENTISSAGE (CFM).

$$\boxed{\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\rho,\, Z,\, X_t\sim p_{t|Z}(\cdot|Z)}\left[\,\|\,u_t(X_t \mid Z) - u_t^\theta(X_t)\,\|^2\right]} \tag{8}$$

En pratique, cette fonction de perte est approximée par lots (*batchs*). L'algorithme pour entraîner un réseau de neurones approximant le champ de vitesses u_t se résume alors à:

- Échantillonner des temps t, de façon uniforme par exemple.
- Échantilloner les variables de conditionnement, par exemple tirer des données $x_1^i \sim q$ depuis notre jeu de données dans le cas où $Z = X_1$.
- Échantillonner la variable X_t depuis $p_{t|Z}(\cdot \mid Z)$, par exemple depuis $\mathcal{N}\left(. \mid tx_1, (1-t)^2 I_d\right)$.
- Calculer $u_t(X_t \mid Z)$, dont nous trouverons l'expression dans la section suivante.
- Calculer la sortie du modèle, $u_t^\theta(X_t)$.
- Calculer l'erreur empirique sur le lot, $\frac{1}{N}\sum_{i=1}^{N} \|u_t(X_t \mid Z) - u_t^\theta(X_t)\|_2^2$.
- Rétropropager les gradients de l'erreur empirique à travers le réseau.

Si les premiers travaux sur le sujet utilisaient $t \sim \mathcal{U}[0,1]$, l'article [26] compare différentes lois pour t et en arrive à la conclusion qu'il vaut souvent mieux utiliser une autre loi, détaillée dans la section .2.8. Dans l'article [54], les auteurs généralisent cette fonction de perte à une classe plus grande: celle des divergences de Bregman. Ils prouvent que la fonction de perte $\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,Z,X_t\sim p_{t|Z}(\cdot|Z)}\left[D_{\text{Breg}}(u_t(X_t|Z), u_t^\theta(X_t))\right]$ possède les mêmes gradients que $\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t,X_t\sim p_t}\left[D(u_t(X_t), u_t^\theta(X_t))\right]$, en ayant l'avantage d'être calculable vu que $u_t(X_t \mid Z)$ admet une forme close. En pratique, nous allons utiliser la fonction de perte (8), qui va se simplifier dans la suite de ce document et possède exactement le même minimiseur que la classe de fonctions de perte introduite par [54].

.2.5 *Flot conditionnel*

RÉSUMÉ. Pour générer un modèle de flot u_t^θ, il nous faut:

- Choisir un chemin de probabilités condtionnel $p_{t|Z}(x|z)$ donnant un chemins de probabilités marginal $p_t(x)$ satisfaisant les conditions à $t = 0$ et $t = 1$.
- Trouver un champ de vitesses conditionnel $u_t(x_t|z)$ générant le chemin de vitesses conditionnel.
- Entraîner le modèle en utilisant la fonctionde perte $\mathcal{L}_{\text{CFM}}$ (8) selon le protocole définit précédemment.

Il nous manque donc à construire $p_{t|Z}(x|z)$ et à déterminer un $u_t(x_t; z)$ générant $p_{t|Z}(x|z)$.

INTERPOLANT. Introduisons un interpolant [1, 2, 54] $\psi_t(x_0, x_1)$ qui relie les états initiaux et finaux, vérifiant donc:

$$\psi_t(x_0, x_1) = \begin{cases} x_0 & t = 0, \\ x_1 & t = 1. \end{cases}$$

Un exemple classique d'interpolant est l'interpolant linéaire, $\psi_t(x_0, x_1) = (1-t)x_0 + tx_1$. Nous allons maintenant construire $p_{t|Z}(x \mid z)$ comme étant l'application de $\psi_t(\cdot, x_1)$ à $p_0(\cdot \mid z)$.

CHEMIN CONDITIONNEL. En supposant que $X_1 \subset Z$ (quitte à échanger X_0 et X_1), on définit

$$p_{t|Z}(\cdot \mid z) = \psi_t(\cdot, x_1)_{\#} p_0(\cdot \mid z), \tag{9}$$

avec

$$p_0(\cdot \mid z) = \begin{cases} \delta_{x_0}(\cdot) & \text{si } X_0 \subset Z, \\ \Pi_{0|Z}(\cdot \mid Z = z) & \text{sinon.} \end{cases}$$

DU POINT DE VUE LAGRANGIEN AU POINT DE VUE EULÉRIEN. Posons $x_{t|z} = \psi_{t|z}(x)$ où x identifie une particule. Nous allons essayer de déterminer le champ de vitesses conditionnel $u_{t|z}$ engendré en conséquence du choix de notre interpolation $\psi_{t|Z}$. Si l'on prend comme analogie la mécanique des fluides, c'est le passage d'un point de vue Lagrangien à un point de vue Eulérien: nous allons déterminer le champ de vitesses (point de vue Eulérien) de nos particules, sachant leur trajectoire (point de vue Lagrangien). Pour ce faire, nous introduisons le changement de variable $x_{t|z} = \psi_{t|z}(x)$ dans (2):

$$\frac{d}{dt}\psi_{t|z}(x) = u_t(\psi_{t|z}(x) \mid z) \tag{10}$$

$$\iff \frac{d}{dt}\psi_{t|z}(\psi_{t|z}(x_{t|z})^{-1}) = u_t(x_{t|z}; z) \tag{11}$$

Ainsi, si nous disposons d'une particule X_t tirée condtionnellement par rapport à Z, nous disposons de la forme close de sa vitesse, $u_t(x_{t|Z}; z) = \dot{\psi}_{t|z}(\psi_{t|z}^{-1}(x_{t|z}))$.
Nous réécrivons donc (8):

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\rho,\, Z,\, X_{t|Z}\sim p_t(\cdot|Z)} \left[\| \dot{\psi}_{t|z}(\psi_{t|z}^{-1}(X_{t|Z})) - u_t^{\theta}(X_{t|Z}) \|^2 \right] \tag{12}$$

Ce faisant, étant donné une interpolation, nous pouvons calculer de façon explicite la fonction de perte à optimiser. Dans la section suivante, nous allons nous intéresser au cas particulier des interpolations affines, et simplifier davantage cette fonction de perte.

.2.6 *Interpolations courantes*

Un grand nombre d'interpolations ont été proposées dans la littérature, s'inspirant souvent des interpolations communément utilisées en diffusion ou s'inspirant du transport optimal. Nous allons en présenter quelques-unes ici.

.2.6.1 *Détours par le transport optimal*

Nous basons en partie cette section sur la section 4.7 de [54], ainsi que sur l'article de *Rectified Flow* [59]. Nous allons essayer de trouver une interpolation utile. Dans ce but, nous introduisons le problème de transport optimal dynamique [5, 76, 101], qui bénéficie d'un fort intérêt dans la communauté du Flow

1 Formellement, pour une mesure μ et une application f, $f_{\#}\mu$ est la loi de $f(X)$ quand $X \sim \mu$.

Matching [48, 53, 54, 58, 59, 77, 98], notamment car il permet une intégration numérique plus rapide et de réduire la variance de l'objectif $\mathcal{L}_{\text{CFM}}$ (due à la variance de $u_t(x_t|Z)$), nulle pour la solution du problème (section 4.2 de [77] ; proposition B.2 de [98]), permettant ainsi d'entraîner des modèles plus rapidement. Soit le problème de transport optimal dynamique, aussi dit de Benamou-Brenier:

$$(p_t^*, u_t^*) = \arg\min_{p_t, u_t} \left\{ \int_0^1 \int_{\mathbb{R}^d} \|u_t(x)\|^2 \, dp_t(x) \, dt \mid \nabla \cdot (p_t u_t) + \frac{\partial p_t}{\partial t} = 0, p_0 = p, p_1 = q \right\} \tag{13}$$

La condition $\nabla \cdot (p_t u_t) + \frac{\partial p_t}{\partial t} = 0$ est l'équation de continuité - c'est la condition nécessaire et suffisante pour que u_t génère p_t, d'après le théorème de conservation de la masse [101]. C'est d'ailleurs en utilisant cette équation que les auteurs de [59] et [54] ont prouvé que notre expression de u_t génère bien p_t.
De façon équivalente, le problème statique de Monge s'écrit

$$\inf_{T:\mathbb{R}^d \to \mathbb{R}^d} \left\{ \int_{\mathbb{R}^d} c(x, T(x)) \, dp(x) \;\middle|\; T_{\#}p = q \right\}. \tag{14}$$

En particulier, pour le coût quadratique $c(x, y) = \|x - y\|^2$, le minimiseur T^* (quand il existe) est la *carte de Brenier* $T^* = \nabla\varphi^*$, gradient d'un potentiel convexe envoyant p sur q [10]. La valeur optimale de (14) pour le coût quadratique coïncide avec $W_2^2(p, q)$ et avec le minimum de la formulation dynamique (13) (aussi dit de Benamou-Brenier) [5, 76, 101].

L'interpolant de McCann [66] associé au problème de Monge est:

$$\psi_t^*(x) = (1 - t)x + t\, T^*(x) \tag{15}$$

Cet interpolant est la solution théorique du problème de Benamou-Brenier (voir [5, 101]). Ainsi:

- Les solutions du problème de transport optimal dynamique sont des trajectoires droites, rapides à intégrer.
- La vitesse le long de cette trajectoire est constante et vaut $T^*(x_0) - x_0$.
- En particulier, cette solution est intégrable en *un seul pas d'Euler*.

En pratique, résoudre le problème de Monge est impossible sur de larges jeux de données, étant donné que la complexité des algorithmes pour le résoudre est cubique en temps et quadratique en mémoire [20, 98, 99].
Cependant, nous savons dorénavant que quand $X_1 = T^*(X_0)$, un flot entraîné avec l'interpolation linéaire $\psi_t(x_0, x_1) = (1 - t) \cdot x_0 + t \cdot x_1$ résoud exactement le problème de transport optimal dynamique. En conséquence, [98] et [77] proposent de calculer le plan de transport optimal sur des batchs, guarantissant d'estimer asyntoptiquement la solution du problème de Benamou-Brenier, arguant que leur méthode facilite l'entraînement des modèles et l'intégration numérique.
Nous pouvons généraliser cet interpolant à une classe plus large, celle des interpolations linéaires :

$$\boxed{\psi_t(x_0, x_1) = \alpha_t x_0 + \beta_t x_1} \tag{16}$$

avec les conditions aux bornes :

$$\alpha_0 = 1, \quad \alpha_1 = 0,$$
$$\beta_0 = 0, \quad \beta_1 = 1.$$

Le couple (α_t, β_t) est appellé *scheduler*. Dans la suite de ce document, nous appellerons *scheduler linéaire* le scheduler $(1 - t, t)$, qui est celui que nous avons pris en exemple précédemment. Nous trouvons dans la littérature les schedulers suivants:

- Linéaire [2, 53, 59], $(1 - t, t)$.

- Cosine [2, 72], $(\cos\left(\frac{\pi}{2}t\right), \sin\left(\frac{\pi}{2}t\right))$.
- Variance Preserving (DDPM adapté): $(\sqrt{1-\exp(-\tilde{t})}, \exp(-\frac{1}{2}\tilde{t}))$ [35, 54], avec $\tilde{t} = \frac{1}{2} \cdot (1-t)^2 \cdot (\beta_{max} - \beta_{min}) + (1-t) \cdot \beta_{min}$, β_{max}, β_{min} deux hyperparamètres.
- Linear Variance Preserving [54]: $(\sqrt{1-t^2}, t)$.
- Polynomial [54]: $(1-t^n, t^n)$.

Le schedule de DDPM [35] appartient à la classe des schedulers dits *Variance-Preserving*, qui vérifient:

$$\begin{aligned}
\forall t \in [0,1], \quad & \mathbb{V}[X_t] = \mathbb{V}[X_0] \\
\iff & \mathbb{V}[\alpha_t X_0 + \beta_t X_1] = \mathbb{V}[X_0] \\
\iff & \alpha_t^2 \mathbb{V}[X_0] + \beta_t^2 \mathbb{V}[X_1] + 2\alpha_t\beta_t \operatorname{Cov}(X_0, X_1) = \mathbb{V}[X_0] \\
\iff & \alpha_t^2 + \beta_t^2 = 1, \quad \text{si } \mathbb{V}[X_0] = \mathbb{V}[X_1] \quad \text{et } \operatorname{Cov}(X_0, X_1) = 0.
\end{aligned}$$

En diffusion, ces schedulers sont très populaires, car les gradients restent dans des ordres de grandeur similaires. C'est aussi le cas en *Flow Matching*. Notons également que le scheduler *Cosine* fait partie des schedulers *Variance-Preserving*.
Si nous utilisons une interpolation linéaire, nous pouvons alors simplifier l'équation de u_t:

$$\boxed{u_t(x) = \mathbb{E}\left[\dot{\alpha}_t X_0 + \dot{\beta}_t X_1 \,\middle|\, X_t = x\right]} \tag{17}$$

Ainsi, si de plus nous utilisons le conditionnement $Z = (X_0, X_1)$, nous pouvons alors simplifier l'équation (12) [54]:

$$\boxed{\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t\sim\rho,\,(X_0,X_1)\sim\Pi_{0,1}}\left[\,\|\,\dot{\alpha}_t X_0 + \dot{\beta}_t X_1 - u_t^\theta(\alpha_t X_0 + \beta_t X_1)\,\|^2\right]} \tag{18}$$

Dans le cas particulier du scheduler linéaire:

$$\mathcal{L}_{CFMLin}(\theta) = \mathbb{E}_{t\sim\rho,\,(X_0,X_1)\sim\Pi_{0,1}}\left[\,\|\,X_1 - X_0 - u_t^\theta(\alpha_t X_0 + \beta_t X_1)\,\|^2\right] \tag{19}$$

En pratique, nous n'avons implémenté que le conditionnement $Z = (X_0, X_1)$ et les interpolations linéaires de la forme $X_t = \alpha_t X_0 + \beta_t X_1$. Nous ne connaissons aucun article utilisant une interpolation non-linéaire.

.2.6.2 *Paramétrisations équivalentes*

En diffusion, plusieurs paramétrisations sont possibles:

- Prédiction du bruit [35].
- Prédiction du signal [72].
- Prédiction de la vitesse [86].

Les mêmes paramétrisations existent dans le cadre du [Flow Matching]. Historiquement, les premiers travaux utilisaient une prédiction en vitesse [2, 53, 59], ce qui a été prouvé comme étant la plus judicieuse pour créer de larges modèles par [26], article fondateur de *Stable Diffusion* 3. Dans les paragraphes suivants, nous allons dériver ces paramétrisations différentes de la vitesse, et étudier leur comportement en $t = 0$ et $t = 1$, ce qui constitue une contribution théorique majeure. En particulier, nous allons découvrir des singularités, et proposer une façon de les réparer.

OBJECTIF ET CHEMIN LINÉAIRE. Nous cherchons à apprendre le champ de vitesses conditionnel défini dans la section précédente 17:

$$u_t(x) = \mathbb{E}\left[\dot{\alpha}_t X_0 + \dot{\beta}_t X_1 \,\middle|\, X_t = x\right]$$

Par linéarité de l'espérance conditionnelle,

$$u_t(x) \;=\; \dot{\alpha}_t\,\mathbb{E}[X_0 \mid X_t = x] \;+\; \dot{\beta}_t\,\mathbb{E}[X_1 \mid X_t = x].$$

PARAMÉTRISATIONS. De l'égalité $x = \alpha_t\,\mathbb{E}[X_0 \mid X_t = x] + \beta_t\,\mathbb{E}[X_1 \mid X_t = x]$, on déduit

$$\mathbb{E}[X_1 \mid X_t = x] = \frac{x}{\beta_t} - \frac{\alpha_t}{\beta_t}\,\mathbb{E}[X_0 \mid X_t = x], \qquad \mathbb{E}[X_0 \mid X_t = x] = \frac{x}{\alpha_t} - \frac{\beta_t}{\alpha_t}\,\mathbb{E}[X_1 \mid X_t = x].$$

Il s'ensuit deux écritures équivalentes de la vitesse :

$$\begin{aligned} u_t(x) &= \frac{\dot{\alpha}_t}{\alpha_t}\,x \;+\; \left[\dot{\beta}_t - \beta_t\,\frac{\dot{\alpha}_t}{\alpha_t}\right]\mathbb{E}[X_1 \mid X_t = x] \\ &= \frac{\dot{\beta}_t}{\beta_t}\,x \;+\; \left[\dot{\alpha}_t - \alpha_t\,\frac{\dot{\beta}_t}{\beta_t}\right]\mathbb{E}[X_0 \mid X_t = x]. \end{aligned}$$

Autrement dit, on peut soit paramétrer directement u_t par un réseau u_t^θ, soit paramétrer $x_{1|t}(x) \coloneqq \mathbb{E}[X_1 \mid X_t = x]$ par $x_{1|t}^\theta(x)$ ou $x_{0|t}(x) \coloneqq \mathbb{E}[X_0 \mid X_t = x]$ par $x_{0|t}^\theta(x)$ et convertir via les identités ci-dessus.

En partant de $X_t = \alpha_t X_0 + \beta_t X_1$ et de la formule de Tweedie [81], qui relie le score $\nabla_x \log p_t(x)$ à l'estimateur $x_{0|t}(x)$ sous l'hypothèse $p = \mathcal{N}(0, I)$, à savoir

$$x_{0|t}(x) = -\,\alpha_t\,\nabla_x \log p_t(x),$$

nous pouvons généraliser à d'autres conversions 1

Depuis \ Vers	u_t	$x_{1\|t}(x_t)$	$x_{0\|t}(x_t)$	$\nabla \log p_t(x_t)$
u_t	u_t			
$x_{1\|t}(x_t)$	$\frac{\dot{\alpha}_t}{\alpha_t}x_t + \frac{\dot{\beta}_t\alpha_t - \dot{\alpha}_t\beta_t}{\alpha_t}x_{1\|t}(x_t)$	$x_{1\|t}(x_t)$		
$x_{0\|t}(x_t)$	$\frac{\dot{\beta}_t}{\beta_t}x_t + \frac{\dot{\alpha}_t\beta_t - \dot{\beta}_t\alpha_t}{\beta_t}x_{0\|t}(x_t)$	$\frac{1}{\beta_t}x_t - \frac{\alpha_t}{\beta_t}x_{0\|t}(x_t)$	$x_{0\|t}(x_t)$	
$\nabla \log p_t(x_t)$	$\frac{\dot{\beta}_t}{\beta_t}x_t - \alpha_t\frac{\dot{\alpha}_t\beta_t - \dot{\beta}_t\alpha_t}{\beta_t}\nabla \log p_t(x_t)$	$\frac{1}{\beta_t}x_t + \frac{\alpha_t^2}{\beta_t}\nabla \log p_t(x_t)$	$-\alpha_t x_{0\|t}(x_t)$	$\nabla \log p_t(x_t)$

Table 1: Conversions entre différentes paramétrisations. Les cases en bleu peuvent facilement être remplies à partir de leur transposée dans le tableau.

SINGULARITÉS APPARENTES. Dans ce nouveau jeu de notations, on a $\alpha_1 = 0$ et $\beta_0 = 0$, ce qui fait apparaître des termes a priori singuliers. En théorie, le champ u_t reste bien défini et continu : la singularité $\frac{\dot{\alpha}_t}{\alpha_t}x_t$ compense exactement le terme $-\,\beta_t\,\frac{\dot{\alpha}_t}{\alpha_t}\,x_{1|t}(x_t)$ en $t = 1$ (lorsque l'on prédit X_1), et la singularité $\frac{\dot{\beta}_t}{\beta_t}x_t$ compense $-\,\alpha_t\,\frac{\dot{\beta}_t}{\beta_t}\,x_{0|t}(x_t)$ en $t = 0$. En pratique toutefois, de légers écarts de prédiction (par exemple $x_{1|t}^\theta(1, x_1) \neq x_1$ ou $x_{0|t}^\theta(0, x_0) \neq x_0$) peuvent empêcher ces compensations numériques et provoquer une explosion de u_t au voisinage de $t = 1$ (si l'on paramètre par $x_{0|t}$) ou de $t = 0$ (si l'on paramètre par $x_{1|t}$).

Le phénomène dépend aussi du solveur ODE : un schéma d'Euler avec peu de pas n'évalue généralement pas u_t exactement en $t = 1$, tandis que certaines méthodes de Runge–Kutta — notamment Dopri5 — évaluent explicitement la fonction en $t = 1$. La singularité en $t = 0$ sous la paramétrisation $x_{1|t}^\theta$

est, elle aussi, problématique. Il est donc utile de traiter ces singularités. À notre connaissance, seul [54] traite de ces singularités - et uniquement pour convertir vers la vélocité, dans le cas particulier où $\Pi_{0,1}(x_0, x_1) = \mathcal{N}(. \mid \mu, \Sigma)\, x_0[0, I]\, q(x_1)$ et $\mathbb{E}[X_1] = 0$. Nous proposons d'autres conversions avec moins d'hypothèses, généralisant ainsi leurs travaux.

.2.6.3 *Traitement des singularités et conversions aux bords*

Nous considérons également les conversions entre représentations (score, prédiction de bruit, $x_{0|t}$, $x_{1|t}$, vitesse). Sauf mention contraire, les formules de score supposent une distribution de départ $\mathcal{N}(0, I_d)$.

Au temps $t = 1$.

- Le score n'est pas défini.
- La vitesse vaut $u_1(x_1) = \dot\beta_1\, x_1 + \dot\alpha_1\, x_{0|1}$. Passer d'une prédiction de $x_{0|t}$ à la vitesse ne pose pas de difficulté, puisque le réseau fournit directement $x_{0|1} = x^\theta_{0|t=1}$. En revanche, pour passer de $x_{1|t}$ à la vitesse, il faut modéliser $x_{0|1} = \mathbb{E}[X_0 \,|\, X_1 = x_1]$. Dans le cas simple où X_0 est indépendant de X_1 et $\mathbb{E}[X_0] = \mu_0$, on a $x_{0|1} = \mu_0$ et

  $$u_1(x_{t=1}) \;=\; \dot\beta_1\, x_{t=1} \;+\; \dot\alpha_1\, \mu_0.$$

  En inpainting, dans le cas où l'on souhaite modéliser la transformation d'une distribution d'images dégradées X_0 par un bruit additif identiquement distribué de moyenne nulle vers la distribution d'images sans altérations X_1 condtionnellement à un masque M (c.f. .2.9 pour les mécanismes de condtionnement), nous pouvons utiliser comme modèle de $x_{0|1}$ l'expression suivante: $\mathbb{E}[X_0 \mid X_1 = x_1,\, M] = X_1 \bigotimes (1 - M)$.
- On ne peut pas, en $t = 1$, déduire $x_{0|t}$ de $x_{1|t}$. En revanche, on peut obtenir $x_{0|1}$ à partir de la vitesse via

  $$x_{0|1} \;=\; \frac{u_1(x_1) - \dot\beta_1\, x_1}{\dot\alpha_1}, \qquad \text{si } \dot\alpha_1 \neq 0.$$
- Évidemment, $x_{1|1} = x_1$.

Table 2: Tableau de conversion au temps $t = 1$

Depuis \ Vers	$\mathbf{u_t}$	$\mathbf{x_{1\mid t=1}}$	$\mathbf{x_{0\mid t=1}}$	**score**
$\mathbf{u_t}$	$u_1(x_1)$	x_1	$\frac{u_1(x_1) - \dot\beta_1 x_1}{\dot\alpha_1}$ (si $\dot\alpha_1 \neq 0$)	
$\mathbf{x_{1\mid t=1}}$	$\dot\beta_1 x_1 + \dot\alpha_1 x_{0\mid1}$ (nécessite $x_{0\mid1}$)	x_1		
$\mathbf{x_{0\mid t=1}}$	$\dot\beta_1 x_1 + \dot\alpha_1 x_{0\mid1}$ (direct)	x_1 (nécessaire)	$x_{0\mid1}$	
score				

Au temps $t = 0$.

- Le score est indépendant de $x_{1|t}$ et vaut simplement $s_0(x_0) = -\dfrac{x_0}{\alpha_0} = -x_0$ (puisque $\alpha_0 = 1$).
- On ne peut pas passer de $x_{0|t}$ à $x_{1|t}$. En revanche,

  $$x_{1|0} \;=\; \frac{u_0(x_0) - \dot\alpha_0\, x_0}{\dot\beta_0}, \qquad \text{si } \dot\beta_0 \neq 0.$$

- Si l'on prédit x_0, on ne peut pas récupérer u_0 sans un modèle pour $\mathbb{E}[X_1 \mid X_0 = x_0]$. C'est plus délicat dans ce sens : en inpainting, ce modèle est généralement inconnu. Si l'on suppose toutefois que $p_{(X_0,X_1)}(x_0, x_1) = \mathcal{N}(x_0 \mid 0, I_d)\, q(x_1)$, alors $\mathbb{E}[X_1 \mid X_0 = x_0] = \mathbb{E}[X_1] =$ constante, ce qui permet de corriger la vitesse. Pour des ponts entre distributions couplées ou avec des *rectified flows*, le problème est plus ardu.

Table 3: Tableau de conversion au temps $t = 0$

Depuis \ Vers	$\mathbf{u_t}$	$\mathbf{x_{1\mid t=0}}$	$\mathbf{x_{0\mid t=0}}$	**score**
$\mathbf{u_t}$	$u_0(x_0)$	$\frac{u_0(x_0)-\dot{\alpha}_0\, x_0}{\dot{\beta}_0}$ (si $\dot{\beta}_0 \neq 0$)	x_0	$-x_0$
$\mathbf{x_{1\mid t=0}}$		$x_{1\mid 0}$	x_0	$-x_0$
$\mathbf{x_{0\mid t=0}}$	Nécessite $\mathbb{E}[X_1 \mid X_0 = x_0]$		x_0	$-x_0$
score	Nécessite $\mathbb{E}[X_1 \mid X_0 = x_0]$		$-s_0 = x_0$	$s_0(x_0) = -x_0$

CONCLUSION PRATIQUE. Paramétrer directement la vitesse u_t^θ est, en pratique, la solution la plus robuste : cela évite les singularités gênantes avec certains solveurs, s'aligne avec la littérature récente, et rejoint les observations empiriques (p.,ex. dans *Scaling Rectified Flow Transformers for High-Resolution Image Synthesis* [26]) selon lesquelles la meilleure configuration consiste à prédire la vitesse directement.

.2.7 *Méthodes d'intégration numérique*

Différents schémas d'intégration numérique sont utilisables pour résoudre l'ODE (1). Parmi ceux-ci, on trouve :

- Euler explicite (utilisé dans [2, 53, 54, 59]).
- Runge-Kutta d'ordre 4 (RK4) (utilisé dans [53]).
- Dormand-Prince (Dopri5) (utilisé dans [54]).
- Midpoint (utilisé dans [54]).

Le choix du schéma d'intégration numérique est une fonction du nombre d'évaluations visés, du scheduler utilisé et du problème. Ainsi, quand le champ de vitesses est proche de celui du transport optimal, un schéma d'Euler explicite est suffisant pour obtenir de bons résultats en très peu d'évaluations - parfois même avec une seule évaluation, comme le prouve InstaFlow [60], dans lequel les auteurs réussissent à générer des images de haute qualité en un seul pas d'Euler grâce à deux rectifications de flot [59] (cf. **??**). En revanche, il est parfois souhaitable d'utiliser des schémas plus précis, notamment quand nous utilisons un scheduler différent du linéaire.

.2.8 *Loi de t*

Bien que dans les premiers travaux [2, 53, 59] le temps était toujours échantillonné uniformément durant l'entraînement, [26] a introduit l'idée d'échantilloner le temps selon une densité plus adaptée. En particulier, ils comparent plusieurs densités et concluent que la meilleure loi pour la génération d'images de haute qualité est une loi *logit-normale*, ie. t suit la loi de $P(Y)$, avec $Y \sim \mathcal{N}(0, I_d)$ et P une fonction logistique de paramètres $(\mu = 0, s = 1)$.

.2.9 *Guidance*

En diffusion, plusieurs méthodes de guidance ont été proposées pour guider la génération vers des échantillons possédant certaines caractéristiques. Les plus connus sont sans doute le conditionnement naïf (ie. concaténer l'information de conditionnement à l'entrée du réseau), le *classifier guidance* [93] et le *classifier-free guidance* [36]. Ces deux dernières méthodes permettent de guider la génération vers des échantillons possédant certaines caractéristiques, même si le modèle n'a pas été entraîné explicitement pour cela. Nous allons brièvement présenter ces méthodes dans le cadre du **FM**.

.2.9.1 *Modèles conditionnels*

La méthode la plus simple pour guider la génération est d'entraîner un modèle conditionnel, ie. un modèle qui prend en entrée une information de conditionnement y (par exemple, une étiquette de classe, un texte, une image, etc.) et qui génère des échantillons en fonction de cette information depuis la distribution $q(x_1 \mid y)$. Cette méthode est simple à mettre en place, mais demande d'avoir un très grand jeu de données pour être efficace, dans lequel plusieurs échantillons partagent un même conditionnement y. En essence, cette méthode revient à remplacer notre modèle $u_t^\theta(x)$ par un modèle conditionnel $u_t^\theta(x, y)$, et à entraîner ce modèle en utilisant:

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\rho,\,(X_0,X_1,Y)\sim\Pi_{0,1,Y}}\left[\,\|\,\dot{\alpha}_t X_0 + \dot{\beta}_t X_1 - u_t^\theta(\alpha_t X_0 + \beta_t X_1, Y)\,\|^2\right] \tag{20}$$

.2.9.2 *Classifier Guidance*

Cette méthode s'appelle Classifier Guidance car elle utilise un modèle de $p_{Y|X_t}(y|X_t = x_t)$. C'est un nom relativement mal choisi, car en pratique n'importe quel modèle de $p_{Y|X_t}$ peut être utilisé, pas forcément un classifieur. Une modélisation utile pour les problèmes inverses est par exemple quand $X_0 \sim \mathcal{N}(0, I_d)$, X_1 est une distribution d'images propres, et Y leur version dégradée. Dans ce cas, il existe des modélisations explicites de $p_{Y|X_t}$ permettant de guider la génération vers des images compatibles avec l'image dégradée $Y = y$ [21].
Elle est basée sur la formule de Bayes:

$$\nabla_{x_t} \log p_{X_t|Y}(x_t|y) = \nabla_{x_t} \log p_{X_t}(x_t) + \nabla_{x_t} \log p_{Y|X_t}(y|x_t) \tag{21}$$

Ainsi, dans le cas où la distribution initiale est une loi normale standard, nous pouvons guider la génération en remplaçant le champ de vitesses $u_t^\theta(x_t)$ par (cf table 1):

$$\begin{aligned} u_t(x_t \mid y) &= a_t x_t + b_t \nabla \log p_{t|Y}(x_t \mid y) && (22)\\ &= a_t x_t + b_t \nabla \log p_{X_t}(x_t) + b_t \nabla \log p_{Y|X_t}(y|x_t) && (23)\\ &= u_t^\theta(x_t) + b_t \nabla \log p_{Y|X_t}(y|x_t) && (24) \end{aligned}$$

En pratique, nous utilisons:

$$\boxed{u_t(x_t \mid y) = u_t^\theta(x_t) + \lambda \nabla_{x_t} \log p_{Y|X_t}(y|x_t)} \tag{25}$$

où $\lambda > 0$ est un hyperparamètre de guidance. $\lambda = 0$ correspond à une génération non guidée, $\lambda = 1$ correspond théoriquement à une génération depuis la distribution conditionnelle $p_{X_1|Y=y}$, et des valeurs plus grandes permettent de renforcer le conditionnement. En pratique, des valeurs entre 1 et 5 sont souvent utilisées. Cette méthode permet de très bons résultats, mais nécessite d'entraîner un modèle supplémentaire dans le cas où $p_{Y|X_t}$ n'est pas connu explicitement.

.2.9.3 *Classifier-free Guidance*

À notre connaissance, la classifier-free guidance (CFG) a été introduite pour la première fois dans [120] dans le cadre du flow matching, directement adaptée de [36]. Le principe est de reformuler l'équation (21) en utilisant un modèle conditionnel $p_{X_t|Y}(x_t|y)$ et un modèle non conditionnel $p_{X_t}(x_t)$:

$$\nabla_{x_t} \log p_{Y|X_t}(y|x_t) = \nabla_{x_t} \log p_{X_t|Y}(x_t|y) - \nabla_{x_t} \log p_{X_t}(x_t) \tag{26}$$

On peut alors réécrire l'équation (22) en utilisant un modèle conditionnel $u_t^\theta(x_t, y)$ et un modèle non conditionnel $u_t^\theta(x_t)$:

$$\begin{aligned} u_t(x_t \mid y) &= u_t^\theta(x_t) + \lambda \left(u_t^\theta(x_t, y) - u_t^\theta(x_t)\right) && (27)\\ &= (1-\lambda) u_t^\theta(x_t) + \lambda u_t^\theta(x_t, y) && (28) \end{aligned}$$

Un seul modèle est en réalité entraîné, en utilisant une probabilité p de retirer le conditionnement y à chaque itération (typiquement, p est choisi entre 0.05 et 0.3). Ainsi, le modèle apprend à la fois à générer des échantillons depuis la distribution non conditionnelle p_{X_1} et depuis la distribution conditionnelle $p_{X_1|Y=y}$. Cette méthode est très populaire en diffusion, car elle permet d'obtenir des résultats de haute qualité sans entraîner de modèle supplémentaire.

.3 LIBRAIRIE IMPLÉMENTÉE

Pour nous doter d'un socle expérimental robuste et renforcer notre maîtrise pratique du cadre de **FM**, nous avons conçu et publié une librairie **FM** dédiée à l'entraînement de modèles de *flow matching* (code). Elle offre une API simple avec un contrôle maximal sur les modèles et leur stratégie d'apprentissage : *schedulers* (linéaire, LinearVP, cosine, polynomial, et adaptation des *diffusers* au **FM**), stratégies d'échantillonnage temporel tirées de [26], guidage par classifieur et *classifier-free guidance*, plusieurs paramétrisations (x_0, x_1, v) ainsi que rectification par *ReFlow* [59]. Cette librairie a servi de base à l'ensemble de nos expériences.

.4 CONCLUSION

Le *Flow Matching* offre un cadre élégant pour apprendre des modèles génératifs en construisant une interpolation continue entre une distribution source p et une distribution cible q. L'idée centrale est de définir d'abord un chemin de probabilités conditionnel $p_{t|Z}$ admettant une forme close, puis d'en dériver un champ de vitesses u_t par marginalisation, et enfin d'apprendre une approximation neuronale u_t^θ minimisant $\mathcal{L}_{\text{CFM}}$.

Nous avons accordé une attention particulière aux interpolations linéaires $X_t = \alpha_t X_0 + \beta_t X_1$ et à leurs différentes paramétrisations (vitesse, signal, bruit, score). L'analyse des singularités en $t = 0$ et $t = 1$ a révélé que la prédiction directe de la vitesse est la plus robuste en pratique, confirmant les observations empiriques de la littérature récente. Les tableaux de conversion proposés facilitent le passage d'une représentation à l'autre, y compris aux temps limites.

Les mécanismes de guidance (*classifier* et *classifier-free*) permettent de conditionner la génération sur des observations externes sans nécessiter de réentraînement complet du modèle. Cette flexibilité est particulièrement précieuse pour les problèmes inverses comme l'inpainting, où l'on souhaite générer des échantillons compatibles avec une image partiellement observée.

La librairie que nous avons développée implémente l'ensemble de ces concepts et servira de base expérimentale pour les chapitres suivants. Nous appliquons maintenant ce cadre théorique au problème d'inpainting d'images, puis de vidéos.

Partie III

INPAINTING D'IMAGES

INPAINTING D'IMAGES

INPAINTING D'IMAGES

Le *Flow Matching*, présenté au chapitre précédent, fournit un cadre théorique puissant et élégant pour l'échantillonnage conditionnel. Nous l'appliquons maintenant à l'inpainting d'images.

Dans ce chapitre, nous allons explorer différentes méthodes pour l'inpainting d'images. Avant de se lancer dans les détails techniques, nous allons clairement définir à quel problème nous faisons référence.

De façon informelle, l'inpainting d'images est le processus de restauration ou de remplissage des parties manquantes ou endommagées d'une image. Cela peut inclure la suppression d'objets indésirables, la réparation de zones corrompues, ou la reconstruction de parties manquantes de l'image. Mais, pour une même image dégradée, plusieurs solutions d'inpainting sont possibles - en ce sens, l'inpainting est un problème mal posé. Il nous faut alors définir un critère de préférence permettant d'évaluer la qualité d'une solution d'inpainting. Un de ces critères peut être la vraisemblance de la complétion : il faut que la génération soit la solution d'un maximum a posteriori (MAP). Mais nous pourrions également viser d'autres critères, comme minimiser l'erreur quadratique moyenne (estimateur MMSE) entre l'image originale et l'image complétée, ou maximiser la qualité perceptuelle.

Nous allons nous intéresser à une modélisation probabiliste de l'inpainting, que nous allons formuler comme une tâche de génération conditionnelle. Nous allons supposer que nous avons une image source $\mathbf{s} \in \mathbb{R}^d$ (avec $d = C \times H \times W$) et une image observée $\mathbf{o} \in \mathbb{R}^d$, avec $\mathbf{o}$ une version dégradée de $\mathbf{s}$ par un masque m valant 1 dans les zones dégradées et 0 ailleurs. Notre objectif est de modéliser la distribution conditionnelle $p(\mathbf{s}|\mathbf{o})$, c'est-à-dire la distribution des images complètes sachant leur version dégradée. En utilisant cette distribution, nous pouvons générer des complétions plausibles pour les parties manquantes de l'image, laissant ainsi l'utilisateur choisir la complétion qui lui semble la plus appropriée.

Il est également à noter qu'à partir de cette modélisation, il est possible d'estimer l'estimateur minimisant l'erreur quadratique moyenne en moyennant plusieurs échantillons issus de la distribution conditionnelle :

$$\hat{\mathbf{s}}_{\text{MMSE}} = \mathbb{E}_{p(\mathbf{s}|\mathbf{o})}[\mathbf{s}] \approx \frac{1}{N} \sum_{i=1}^{N} \mathbf{s}_i, \quad \mathbf{s}_i \sim p(\mathbf{s}|\mathbf{o})$$

De plus, les modèles que nous allons développer peuvent potentiellement être insérés dans un cadre *Plug-and-Play* pour obtenir un estimateur du maximum a posteriori. Cependant, nous ne nous attarderons pas sur cette approche dans ce rapport. Pour une revue de littérature sur les méthodes de Plug-and-Play, nous invitons le lecteur intéressé à consulter les articles [42] et [21].

Nous présentons d'abord un état de l'art des méthodes d'inpainting, puis développons deux approches basées sur le *Flow Matching* : une première utilisant la *classifier-free guidance*, une seconde exploitant les propriétés du transport optimal.

.1 ÉTAT DE L'ART

Nous allons maintenant passer en revue les méthodes d'inpainting d'images, en nous concentrant principalement sur les approches récentes basées sur des modèles profonds et des modélisations probabilistes. L'inpainting d'images est un domaine de recherche actif, avec de nombreuses techniques développées au fil des ans, et ce depuis les années 2000. Pour une analyse plus poussée de la littérature des méthodes d'inpainting par deep learning, nous recommandons la lecture des articles [78, 109]. Pour une revue des méthodes n'utilisant pas d'apprentissage profond, nous renvoyons le lecteur vers les articles [32, 41, 71]. Il est à noter que les méthodes probabilistes basées sur des modèles profonds sont relativement récentes et n'incluent pas la majorité des méthodologies d'inpainting, développées depuis les années 2000. Les

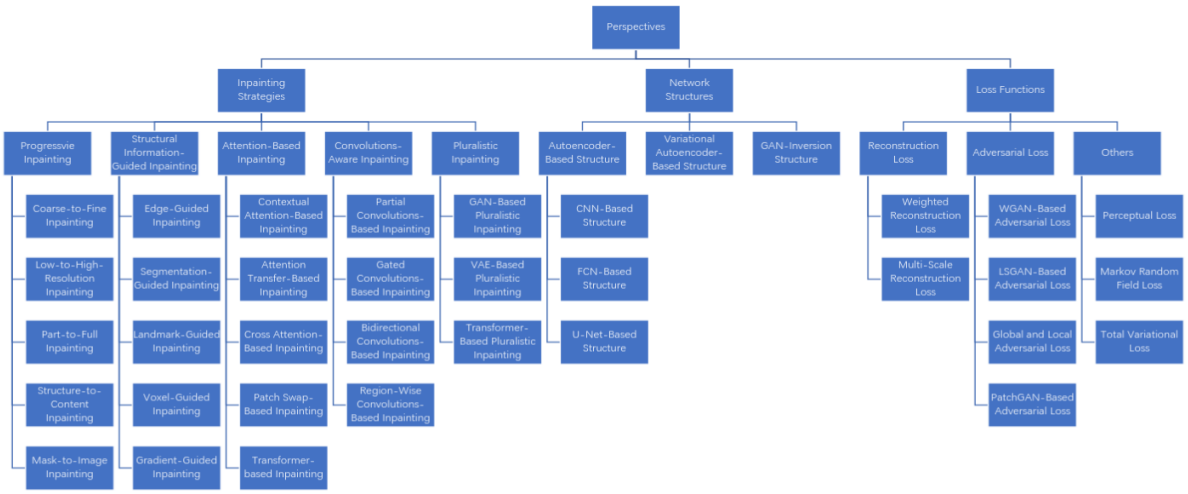


Figure 1: Catégorisation des méthodes d'inpainting d'images. Source: [109]

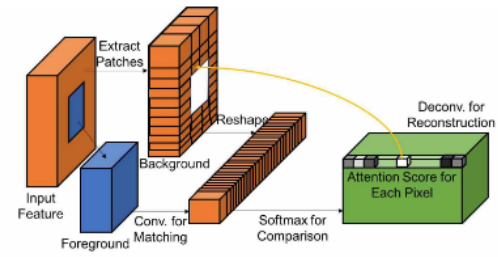


Figure 2: Calcul de similarité. Source: [113]

pistes de recherche en deep learning sont nombreuses, et nous allons les séparer en deux catégories principales: l'amélioration des architectures (introductions de nouvelles couches, méthodes "coarse-to-fine", améliorations progressives, etc.) et l'amélioration des modélisations (fonctions de pertes, modélisation probabiliste, etc.).

.1.1 *Architectures*

.1.1.1 *Méchanismes d'attention*

Les méthodes d'inpainting profond basées sur des méchanismes d'attention ont très rapidement gagné gagné en popularité, notamment car c'est la continuation logique des méthodes par patchs et car cela permet de capturer des dépendances à longue portée dans les images. Ainsi, en 2018 l'article [106] adapte directement l'algorithme de Non-Local Means (NLM) [11] en une opération non-locale utilisée dans un réseau de neurones, permettant ainsi d'exploiter de l'information spatialement distante et démontrant l'intérêt des méthodes non locales appliquées aux réseaux de neurones.

ATTENTION CONTEXTUELLE L'attention contextuelle [113] permet de mieux capturer les relations entre les pixels en tenant compte du contexte global de l'image. Cette approche a été particulièrement efficace pour l'inpainting, car elle permet de générer des complétions plus cohérentes et réalistes en exploitant les informations contextuelles disponibles dans l'image. Cettte méthode se décompose en deux parties:

- *Calcul de la similarité*: Les auteurs extraient les patchs (3×3) des features de la partie inconnue de l'image, et les comparent avec les patchs de la partie connue en utilisant le produit scalaire. Cette opération est équivalente à une convolution, où les patchs de la partie connue servent de filtres

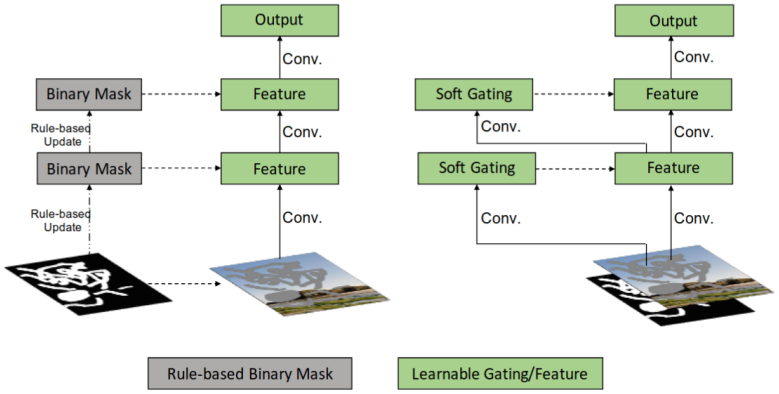


Figure 3: À gauche: convolution partielle; À droite: Gated Convolution. Source: [114]

convolutifs. Une opération de *softmax* par *channel* est ensuite effectuée. Le résultat est une carte de similarité qui indique à quel point chaque patch inconnu est similaire aux patchs connus.

- *Reconstruction*: La reconstruction de la partie inconnue de l'image est effectuée en utilisant une opération de convolution transposée (deconvolution) avec les patchs connus comme filtres, pondérée par la carte de similarité obtenue précédemment. Cela permet de combiner les informations des patchs connus pour générer une complétion cohérente de la partie inconnue.

Il est à noter que le coût algorithmique de cette méthode est élevé, en $O(N^2)$, où N est le nombre de pixels dans l'image. C'est le cas de la majorité des méthodes basées sur l'attention. [84] prouve qu'il est également possible d'utiliser la distance Euclidienne au lieu du produit scalaire pour le calcul de similarité.

TRANSFORMERS De par leurs capacités à modéliser des dépendances à longue portée et le succès de leur mise à l'échelle, les transformers [100] se sont énormément popularisés dans le domaine de la vision par ordinateur. En 2021, l'article [103] ajoute un module transformer dans un réseau d'inpainting grossier afin de mieux utiliser le contexte global de l'image, suivi par un module de raffinement (coarse-to-fine), qui bénéficie ainsi de la guidance du résultat grossier. En 2022, l'article [51] propose une architecture de transformer dite *Mask-Aware Transformer* (MAT).

Bien d'autres architectures basées sur des transfomers ont été proposées, telles que [12, 38, 44] se basant sur des vision transformers (ViT) [23], [22] utilisant un Unet-transformer, [68] utilisant des *Swin Transformers* [61], etc. Pour une revue plus complète des architectures basées sur des transformers, nous invitons le lecteur à consulter [25], qui analyse spécifiquement les méthodes d'inpainting image et vidéo basées sur des transformers.

.1.1.2 *Convolutions*

Les convolutions sont la brique de base des réseaux de neurones convolutifs (CNN), qui sont largement utilisés pour diverses tâches de vision par ordinateur, y compris l'inpainting d'images. Cependant, les filtres de convolutions traditionnels ne sont pas particulièrement adaptés pour gérer les zones manquantes dans les images. Pour remédier à cela, plusieurs variantes de convolutions ont été proposées.

CONVOLUTIONS PARTIELLES Une couche de convolutions partielles [56] consiste en une opération de convolution ne considérant que les voisins non-masqués du noyau et une fonction de mise à jour du masque, le dilatant progressivement. Cela permet théoriquement d'empêcher la propagation d'informations provenant des zones masquées.

GATED CONVOLUTIONS Les *Gated convolutions* [114] sont l'extension logique des convolutions partielles: plutôt que d'utiliser une règle dure pour la mise à jour du masque de propagation, une *porte*

est apprise grâce à une simple opération de convolution suivie d'une sigmoïde, permettant de moduler la propagation des informations de façon plus flexible. [112] proposent une variante de cette méthode, réduisant le nombre de paramètres.

FOURIER CONVOLUTIONS C'est une approche radicalement différente des précédentes - elle est basée sur le théorème de convolution en Fourier, qui dit qu'un produit scalaire dans le domaine de Fourier est équivalent à une convolution dans le domaine image. L'article [18] propose alors d'utiliser le contexte global dans les premières couches du modèle en faisant une transformée de Fourier rapide par *channel* et en combinant une branche locale utilisant des convolutions conventionnelles et une branche globale. La branche globale applique une fonction *Real FFT2d* au tenseur d'entrée, concatène les parties réelle et imaginaire, puis applique une convolution 1×1 avant de faire une transformée de Fourier inverse. Cette méthode est particulièrement efficace pour capturer des informations à longue portée dans les images, ce qui a été utilisé par les auteurs de Large Mask inpainting (LaMa) [95] pour l'inpainting d'images. Leur modèle généralise particulièrement bien aux dimensions d'images non vues lors de l'entraînement, pouvant atteindre jusqu'à 2048×2048.

.1.2 *Méthodes probabilistes*

Les méthodes probabilistes ont également beaucoup été explorées dans la littérature d'inpainting d'image pluraliste. Ainsi, les Variational Autoencoders (VAE) [46] ont d'abord été utilisés par Zheng et al. [119] en 2019 utilise une modification d'un CVAE avec deux branches - une branche dite de reconstruction qui modélise le prior des parties manquantes et reconstruit l'image complète depuis cette distribution. La branche générative infère la variable latente condtionnelle des zones masquées. [33] s'intéresse à l'inpainting d'images de fashion. Ils séparent la tâche en deux étapes: d'abord, un VAE génère une segmentation condtionnellement à l'image dégradée, puis un second VAE génère l'image complète condtionnellement à la segmentation et à l'image dégradée. [33] montre comment il est possible d'utiliser un VAE préentraîné pour faire de l'inpainting - en encodant directement l'image dégradée et en échantillonnant dans la distribution latente.

Les méthodes par *GAN* (Generative Adversarial Networks) [29] ont également été largement utilisées [57, 118]. Il est à noter qu'il existe beaucoup d'approches fonctionnant par inversion de GANs [108], qui consiste à essayer de trouver un code latent qui, décodé, produise la partie observée de l'image, et interpole naturellement les zones à remplir. Dans sa forme la plus simple, l'inversion de GANs consiste à résoudre le problème d'optimisation suivant (avec z un bruit ou vecteur de l'espace latent du GAN):

$$z^* = \arg\min_z \|m \odot (G(z) - \mathbf{o})\|_2^2 + \lambda R(z)$$

Mais toute une variété d'autres fonctions de ressemblance ont été testées. Bien que ces approches soient très populaires, elles reposent non pas sur une modélisation probabiliste, mais sur les hypothèses suivantes:

- Un espace latent structuré et continu permettant de faire une descente de gradient sur la fonction de ressemblance.
- Un espace latent couvrant l'ensemble des images possibles.
- Bijectivité entre l'espace latent et l'espace des images - à une image un unique code et à un code une unique image
- Cohérence sémantique entre les parties connues et inconnues de l'image.
- Hypothèse de déterminisme - pour une image donnée, il n'existe qu'une seule complétion correcte.

Certains travaux ont également exploré l'utilisation de *Normalizing Flows* [80], notamment [104] qui, inspiré par Glow [45] et son extension conditionnelle [63], propose d'utiliser un Normalizing Flow condtionnel pour apprendre la distribution des structures, avant qu'un autre générateur utilise ce résultat

pour générer l'image complète (structure-then-texture). Cependant, l'intérêt de la communauté s'est vite détourné des Normalizing Flows, peu adaptés aux problèmes en très hautes dimensions, au profit des modèles de diffusion. Parmi ces méthodes, nous pouvons notamment citer RePaint [64], qui permet d'utiliser un modèle pré-entraîné à la génération pour faire de l'inpainting, et Palette [85] qui présente un cadre *image-to-image* pour la génération conditionnelle par diffusion. Cependant, ces méthodes sont coûteuses en temps de calcul, nécessitant plusieurs centaines d'itérations pour générer une image. Pour remédier à cela, des approches plus récentes proposent d'utiliser des modèles de diffusion latente (LDM) [82] pour l'inpainting, permettant ainsi de réduire considérablement le coût computationnel tout en maintenant une qualité de génération élevée. Il est également possible de prendre de larges modèles pré-entraînés comme *Stable Diffusion 3* et de les spécialiser pour l'inpainting en utilisant un *ControlNet* [117], permettant d'adapter d'adapter les modèles text-to-image (T2I) en rajoutant d'autres conditionnements, basés sur des images (profondeur, image incomplète, canny, ...).

.2 PROTOCOLE EXPÉRIMENTAL

Nous adoptons la méthodologie d'*internal learning* de Cherel et al. [17], consistant à entraîner le modèle directement sur l'image à restaurer, évitant tout biais lié à des données externes. Nous présentons d'abord le protocole commun aux deux méthodes développées, puis détaillons chaque approche.

.2.1 *Internal learning et acquisition des données*

.2.1.1 *Principe*

L'*internal learning* exploite uniquement l'information contenue dans l'image de test $x_{\text{test}} \in \mathbb{R}^{C\times H\times W}$ pour entraîner le modèle. Disposant d'une région à restaurer définie par le masque M_{test}, nous extrayons aléatoirement des patchs d'entraînement $x_{\text{train}} \in \mathbb{R}^{C\times 256\times 256}$ uniquement depuis les zones *non masquées*. Cette stratégie garantit que le modèle apprend la distribution de texture spécifique à l'image sans jamais observer la région à compléter.

.2.1.2 *Pipeline d'entraînement*

La Figure 5 illustre le pipeline complet. Un patch cible X_1 est extrait aléatoirement des zones non masquées (rectangle vert pointillé), en évitant soigneusement la zone de test M_{test} (rouge). Pour chaque patch, un masque d'entraînement M_{train} est généré via l'algorithme de Yu et al. (2019) [114], qui simule des coups d'effaceur aléatoires en tirant une succession de traits en directions inverses, comme décrit dans la figure 4. Ce masque est appliqué pour créer le patch dégradé $\tilde{X} = X_1 \otimes (1 - M_{\text{train}}) + Z \otimes M_{\text{train}}$, où $Z \sim \mathcal{N}(0, I)$.

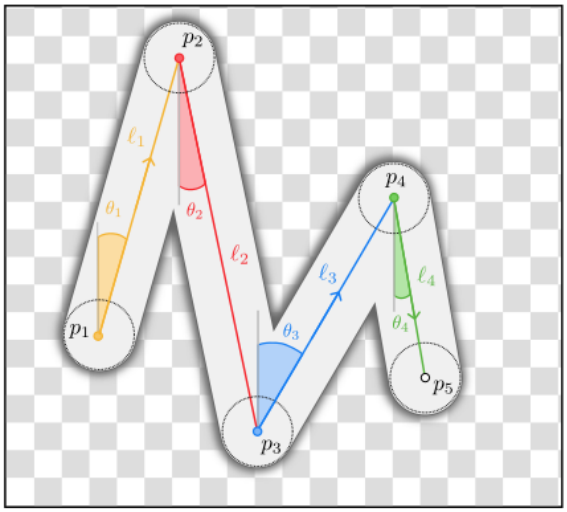
Dans la méthode A, nous définissons p comme étant une loi normale standard, et formulons une tâche génération conditionnelle: nous allons utiliser la méthode de classifier-free guidance pour échantillonner depuis $q(\cdot \mid \tilde{X})$. L'interpolation linéaire $X_t = (1-t)X_0 + tX_1$ produit alors une transition entre un bruit pur et un patch complet. C'est presque la même méthodologie que la majorité des modèles de diffusion *text-to-image*, à l'exception que le conditionnement est maintenant une image.

Dans la méthode B, nous définissons p comme étant la distribution des patchs dégradés $\tilde{X}$, et ainsi le processus d'interpolation linéaire $X_t = (1-t)X_0 + tX_1$ produit une transition graduelle entre l'état dégradé ($t = 0$) et l'état cible ($t = 1$). Le modèle apprend à générer cette transition en prédisant le champ de vitesses $u_t^\theta(X_t, M_{\text{train}})$.

.2.1.3 *Prétraitement*

Les patchs sont normalisés indépendamment par canal pour avoir une moyenne nulle et une variance unitaire. Pour chaque canal $c \in \{R, G, B\}$, la normalisation s'écrit :

$$\tilde{x}_c = \frac{x_c - \mu_c}{\sigma_c},$$



$p \sim \mathcal{U}(0, W) \times \mathcal{U}(0, H)$
for $i = 1, ..., N$ **do**
  $\theta \sim \mathcal{U}(0, \theta_{\max})$
  if i % 2 == 0 **then**
    $\theta \leftarrow 2\pi - \theta$ ▷ reverse mode
  end if
  $\ell \sim \mathcal{U}(0, \ell_{\max})$
  $p' \leftarrow p + \ell \cdot (\cos(\theta)\ \sin(\theta))^T$
  Draw line from p to p'
  $p \leftarrow p'$
end for

Figure 4: Méthodologie de Yu et al. [114] pour la création de masques imitant un utilisateur faisant usage d'un effaceur. Figure extraite de leur article.

où les statistiques $\mu_c = \mathbb{E}[x_c]$ et $\sigma_c = \sqrt{\mathbb{V}[x_c]}$ sont calculées sur l'ensemble des pixels des zones non masquées de x_{test}. Cette normalisation stabilise l'entraînement en centrant les gradients et accélère la convergence. Nous utilisons une banque de 15 000 masques pré-générés, échantillonnés uniformément durant l'entraînement, avec des couvertures variant entre 15% et 50% de la surface du patch.

.2.2 *Architecture commune*

Les deux méthodes reposent sur une architecture U-Net, inspirée de Cherel et al. [17] (voir Figure 6). Celle-ci comporte quatre niveaux de résolution, chacun composé de deux blocs convolutionnels suivis d'une normalisation (BatchNorm) et d'une activation non linéaire (ReLU). Le nombre de canaux est fixé à 32 sur l'ensemble des niveaux afin de maintenir une empreinte mémoire réduite et éviter le surapprentissage. Les transitions entre les niveaux sont assurées par un opérateur de max-pooling lors de la descente (encodeur) et par un upsampling bilinéaire lors de la remontée (décodeur), avec des connexions de type *skip* permettant de préserver l'information spatiale fine.

Le temps de diffusion t est encodé à l'aide de *Fourier Features* [96]. Ce vecteur est ensuite passé à travers une couche fully-connected, puis injecté dans le réseau par concaténation avec les autres entrées du modèle, x_t, M, et potentiellement $\tilde{X}$.

.2.3 *Hyperparamètres communs*

Les deux modèles sont entraînés pendant 9 400 itérations avec un batch size de 128 sur une NVIDIA A100 40Go. Une taille de batch relativement conséquente a été choisie en raison de la variance de l'objectif à minimiser, et nous avons choisi 9 400 itérations car c'était empiriquement le nombre d'itérations nécessaire pour que la fonction de perte cesse de diminuer. L'optimisation utilise AdamW [62] avec un taux d'apprentissage de 3×10^{-4}. L'entraînement prend environ 30 minutes. L'échantillonnage du temps suit $t \sim \text{LogitNormal}(\mu = 0, s = 1)$, accordément aux études menées par [26]. L'intégration du champ de vitesses utilise Euler explicite avec 100 pas.

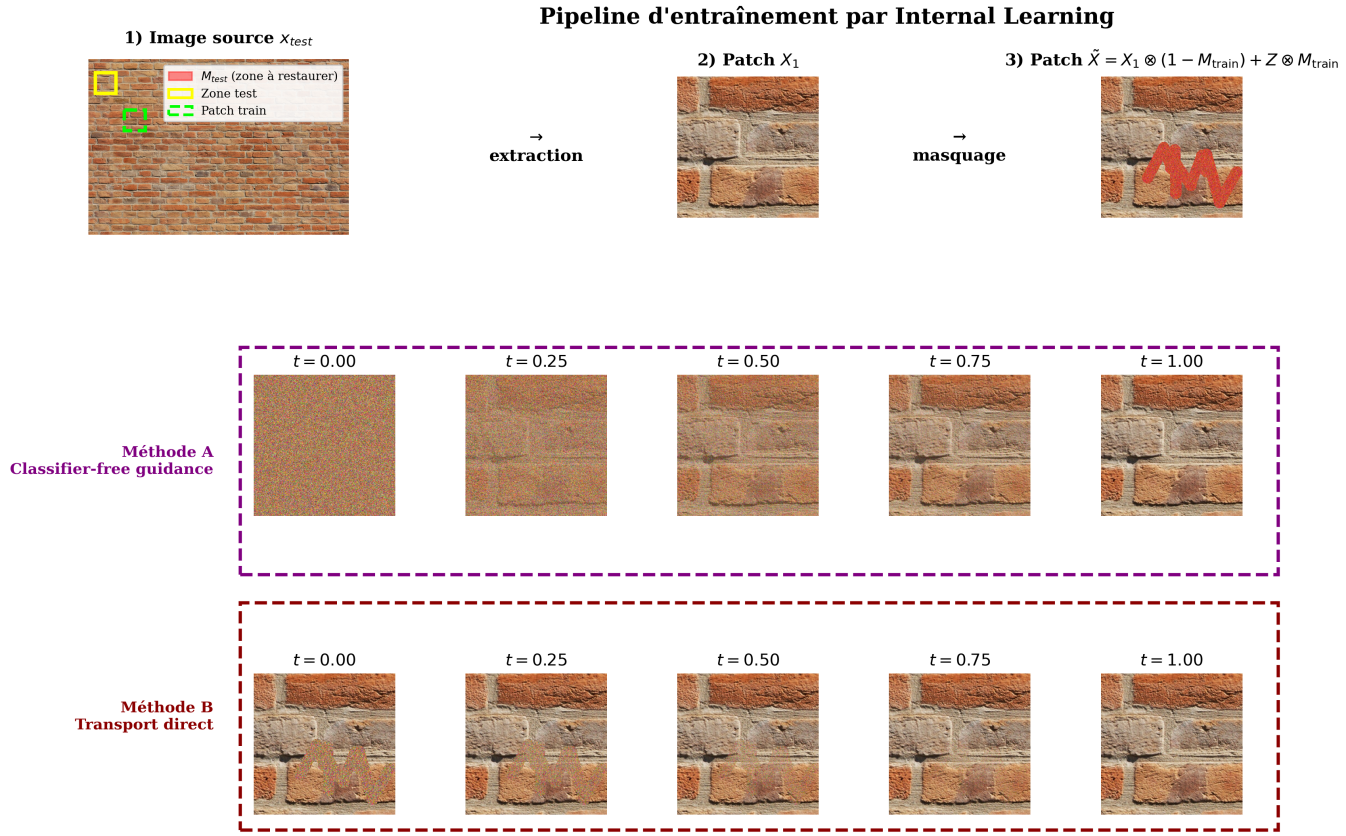


Figure 5: Pipeline d'entraînement par *internal learning*. **(Haut)** Extraction d'un patch X_1 depuis les zones non masquées de x_{test} (vert), puis application d'un masque aléatoire M_{train} pour créer le patch dégradé $\tilde{X}$ (rouge). **(Milieu)** Interpolation linéaire montrant la transition progressive pour cinq valeurs de $t \in [0, 1]$ pour la méthode A, dans laquelle la distribution d'origine est une loi normale standard. **(Bas)** Même interpolation linéaire mais pour la méthode B, dans laquelle la distribution d'origine est la version dégradée des patchs.

.3 MÉTHODE A: CLASSIFIER-FREE GUIDANCE

La première méthode que nous avons testée repose sur l'utilisation de la *classifier-free guidance*. Notre modélisation est la suivante:

- **Distribution d'origine:** $X_0 \sim \mathcal{N}(0, I)$
- **Conditionnement:** $Y = (\tilde{X}, M) \sim p_y(Y)$
- **Distribution cible:** $X_1 \sim p_{data}(\cdot|Y)$
- **Interpolation:** $X_t = (1-t)X_0 + tX_1$
- **Classifier-free guidance:** $u_t(x_t \mid y) = (1-\lambda)u_t^\theta(x_t) + \lambda u_t^\theta(x_t, y)$ (27)
- **Entrées du modèle:** (x_t, t, y) ou (x_t, t) avec probabilité p_{drop}. $y = \varnothing$ est remplacé par un tenseur appris lors de l'entraînement, de la même dimension que Y.
- **Sortie du modèle:** $u_t^\theta(x_t, y)$ avec probabilité $1 - p_{drop}$ ou $u_t^\theta(x_t)$ avec probabilité p_{drop}.
- **Échantillonage du temps lors de l'entraînement**: $t \sim \text{LogitNormal}(\mu = 0, s = 1) \leftrightarrow t = \text{sigmoid}(Z; \mu, s)$, avec $Z \sim \mathcal{N}(0, 1)$, conformément à l'analyse de [26].
- **Fonction de perte:**

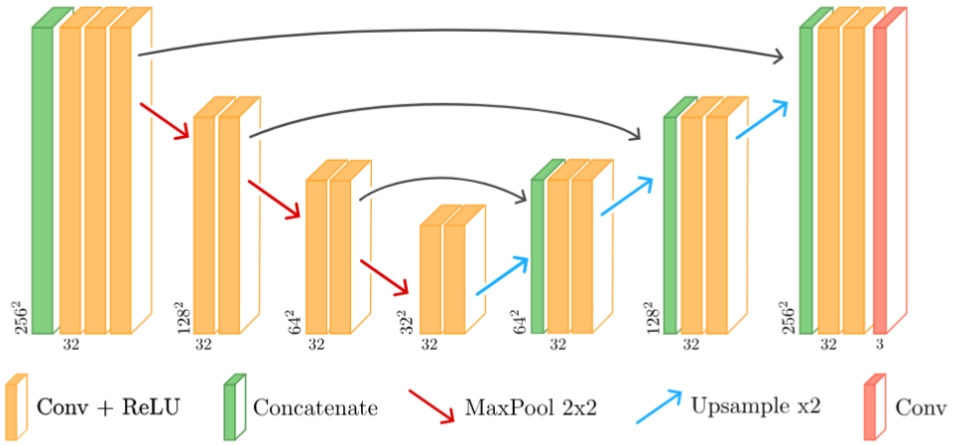


Figure 6: Architecture Unet utilisée pour les deux méthodes. Source: [17]

$$\mathcal{L}_{\text{CFG}}(\theta) = \mathbb{E}_{t,\, x_0 \sim p_0,\, x_1 \sim p_1,\, y \sim p_y(\cdot|x_1),\, \epsilon \sim \text{Bernoulli}(p_{\text{drop}})} \Bigg[\tag{29}$$

$$\left\| (x_1 - x_0 - u_t^\theta\left((1-t)\cdot x_0 + t \cdot x_1 \mid \tilde{y}\right)) \otimes M_{\text{train}} \otimes (1 - M_{\text{test}}) \right\|_2^2 \Bigg], \tag{30}$$

avec:

$$\tilde{y} = \begin{cases} y, & \text{si } \epsilon = 0, \\ \varnothing, & \text{si } \epsilon = 1. \end{cases}$$

En pratique toutefois, la fonction de perte et le processus d'entraînement sont totalement gérés par la librairie que nous avons développé, nous permettant de nous abstraire en bonne partie des soucis d'implémentation. Notez l'utilisation de M_{test} dans la fonction de perte, qui permet de ne pas entraîner le modèle sur la zone à remplir de l'image de test. Le terme M_{train} est le masque aléatoire généré pour l'image d'entraînement x_{train}. Le modèle n'est donc entraîné que pour remplir la zone masquée de l'image d'entraînement, en ignorant la zone masquée de l'image de test et le reste de l'image d'entraînement.

.3.1 *Échantillonnage : stabilisation par réinjection du contexte*

Lors de l'inférence, nous appliquons l'astuce de *RePaint* [64], consistant à réinjecter à chaque pas la partie connue de l'image observée. Formellement, nous remplaçons la prédiction du champ de vitesses par :

$$\tilde{u}_t^\theta(x_t, Y) = u_t^\theta(x_t, Y) \otimes m + (x_1 - x_0) \otimes (1 - m),$$

avec $x_1 - x_0$ le champ de vitesse conditionnel théorique $u_t(x_t|y)$ de l'image pour le scheduler linéaire, ce qui empêche le modèle de modifier les zones déjà observées et stabilise la reconstruction.
D'autres schedulers peuvent être utilisés, mais d'après nos expériences le scheduler linéaire est celui qui donne les meilleurs résultats (ce qui est là encore conforme à l'analyse de [26]).

.3.2 *Méthode d'intégration*

La méthode d'intégration du champ de vitesse peut être choisie après l'entraînement du modèle. En pratique, nous utilisons la méthode d'Euler explicite, qui est simple et efficace, avec un pas de temps $\Delta t = 1/100$, qui produit des résultats de bonne qualité en un temps raisonnable.

.3.3 *Compromis diversité - fidélité*

L'utilisation de la *classifier-free guidance* permet de contrôler le compromis fidélité avec la référence / diversité en ajustant le paramètre λ. En pratique, nous utilisons une valeur de λ de 1.1, qui offre un bon équilibre entre la qualité des complétions et la diversité des échantillons générés. Une valeur de λ nulle entraîne des complétions sémantiquement incorrectes, tandis que des valeurs trop élevées fait apparaître des artefacts aux bords du masque. Voir la figure 9 en annexe.

.4 MÉTHODE B: FLOT ET TRANSPORT OPTIMAL

Nous formulons désormais le problème de la manière suivante : il s'agit de transporter une distribution Π_0 d'images corrompues vers une distribution Π_1 d'images propres.
Sur le plan théorique, le *flow matching* garantit uniquement que les échantillons issus de Π_0 sont transportés vers la distribution Π_1, sans assurer pour autant que ce transport soit optimal ni que l'on échantillonne depuis $\Pi_{1|0}(\cdot \mid X_0)$.

Toutefois, le *flow matching* présente une propriété remarquable mise en évidence par *Rectified Flow* [59]. En notant Z_1 la distribution générée par le modèle et c un coût convexe (par exemple la distance quadratique), on dispose de l'inégalité

$$\mathbb{E}\big[c(Z_1 - X_0)\big] \leqslant \mathbb{E}\big[c(X_1 - X_0)\big].$$

Autrement dit, en moyenne, les échantillons générés sont plus proches des points de départ que ne le sont les couples de données initialement observés. En particulier, si

$$\mathbb{E}\big[c(Z_1 - X_0)\big] = \mathbb{E}\big[c(X_1 - X_0)\big],$$

alors les échantillons générés sont, en moyenne, aussi proches des images corrompues que les images propres le sont elles-mêmes des corrompues. Cette propriété est notable, car elle suggère que le modèle conserve une forte fidélité visuelle par rapport aux données d'entrée, tout en les rapprochant de la distribution cible.
Nous réutilisons donc la méthodologie précéndente, en modifiant:

- **Distribution d'origine:** $X_0 = X_1 \otimes (1 - M_{\text{train}}) + Z \otimes M_{\text{train}}$, $Z \sim \mathcal{N}(0, I)$.
- **Conditionnement:** Masque $M = M_{\text{train}} \cup M_{\text{test}}$.
- **Distribution cible:** $X_1 \sim p_{\text{data}}$
- **Interpolation:** $X_t = (1-t)X_0 + tX_1$
- **Aucune guidance** - on se contente de donner le masque et x_t en entrée du modèle.
- **Entrée du modèle:** (x_t, t, M)
- **Sortie du modèle:** $u_t^\theta(x_t)$.
- **Échantillonage du temps lors de l'entraînement**: Même qu'auparavant, $t \sim \text{LogitNormal}(\mu = 0, s = 1)$.
- **Fonction de perte:**

$$\mathcal{L}_{\text{Flow}}(\theta) \;=\; \mathbb{E}_{t,\, x_1 \sim p_{\text{data}},\, M_{\text{train}} \sim p_{\text{train}},\, z \sim \mathcal{N}(0,I)} \left[\left\| \big((x_1 - x_0) - u_t^\theta(x_t, Y)\big) \otimes M_{\text{train}} \otimes (1 - M_{\text{test}}) \right\|_2^2 \right],$$

avec

$$x_0 = x_1 \otimes (1 - M_{\text{train}} \cup M_{\text{test}}) + z \otimes M_{\text{train}} \cup M_{\text{test}}, \quad x_t = (1-t)x_0 + tx_1.$$

Nous utilisons la même astuce que précédemment, en remplaçant $u\_t^\theta(x\_t, M)$ par

$$\tilde{u}_t^\theta(x_t, M) = u_t^\theta(x_t, M) \otimes M \; + \; (x_1 - x_0) \otimes (1 - M),$$

où $(x_1 - x_0)$ représente le champ de vitesse conditionnel théorique associé au scheduler linéaire.

.5 RÉSULTATS

Nous présentons ici les résultats des deux approches d'inpainting développées dans ce travail. Les résultats qualitatifs permettent d'évaluer la cohérence des textures reconstruites, la continuité des structures globales, ainsi que la stabilité du processus d'échantillonnage.

.5.1 *Méthode A : Inpainting par Flow Matching avec Classifier-Free Guidance*

La Figure 7 illustre la qualité des reconstructions obtenues pour la première méthode. Les complétions produites sont globalement corrrectes, l'intérieur de la complétion étant par lui-même vraisemblable, et relativement fidèle à l'extérieur. Nous constatons toutefois certains défauts de raccord aux bords.

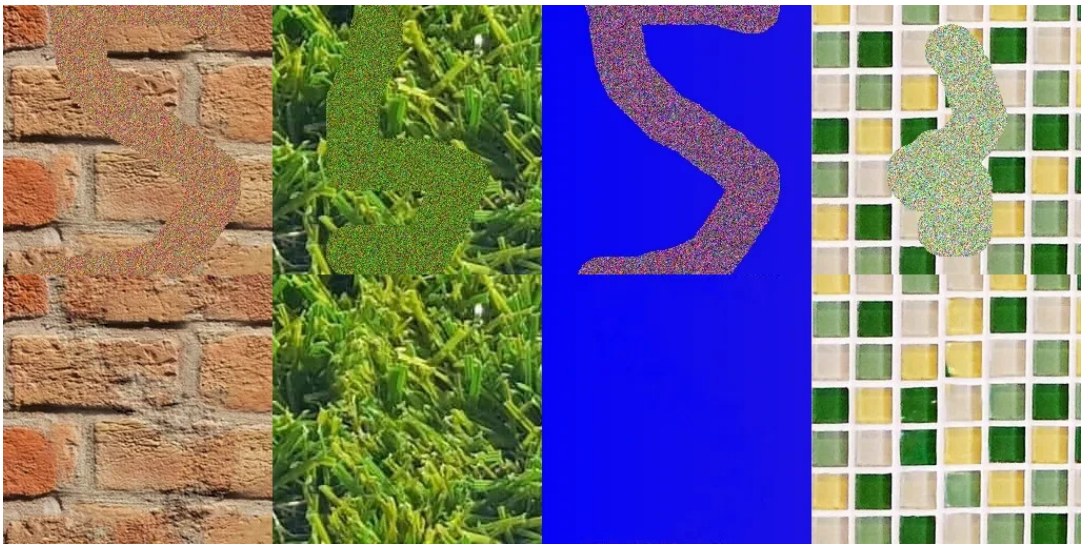


Figure 7: Résultats qualitatifs de la première méthode. Ligne du haut : image dégradée. Ligne du bas : complétion générée. Inférence avec 100 pas d'Euler et $\lambda = 1.1$.

IMPACT DU NOMBRE DE PAS D'INTÉGRATION. La Figure 8 montre l'effet du nombre de pas d'Euler lors de l'inférence. Grâce à la nature déterministe du *Flow Matching*, environ 20 pas suffisent pour obtenir une complétion de bonne qualité, ce qui rend l'approche extrêmement efficace en temps d'inférence.

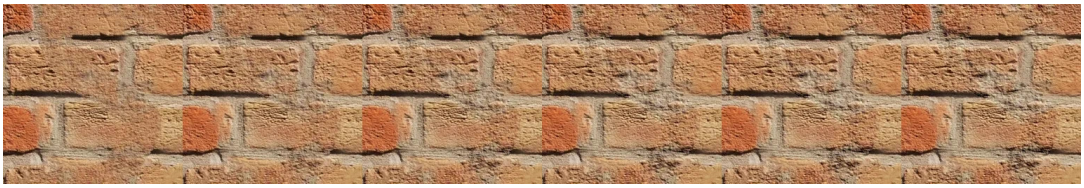


Figure 8: Influence du nombre de pas d'intégration n_{steps}. De gauche à droite : $5, 10, 20, 50, 100, 1000$ pas. Ici, $\lambda = 1.1$.

IMPACT DU PARAMÈTRE DE GUIDANCE λ. La Figure 9 illustre la manière dont λ contrôle le compromis entre fidélité au contexte observé (λ faible) et cohérence globale/stylisation (λ élevé). Nous observons en effet que quand $\lambda = 0$, le modèle fait de la génération non guidée. Pour $\lambda = 1$, la classifier-free guidance guarantit théoriquement d'échantillonner depuis la distribution conditionnelle, et c'est bien ce que l'on voit. Pour λ plus grand, nous observons par contre des effets aux bords. Nous attribuons ces effets à la variance plus grande dépendance du débruitage dans cet zone par rapport à la condition extérieure. Ainsi, dans ces zones il existe une grande différence entre le débruitage conditionné à l'extérieur et le débruitage non conditionné, et l'utilisation d'un trop gros facteur vient amplifier davantage cette différence de prédiction.

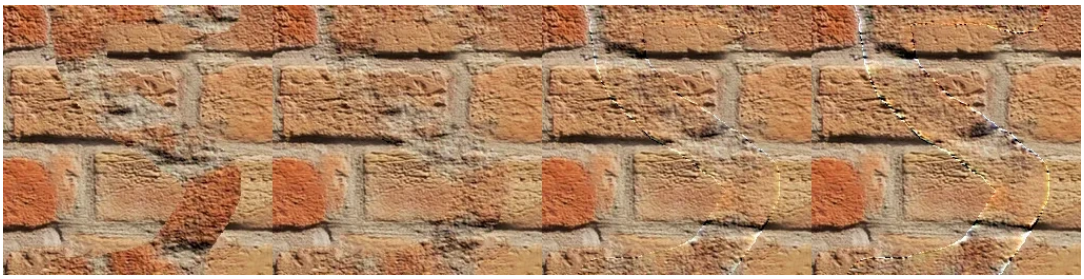


Figure 9: Effet du paramètre de guidance λ. De gauche à droite : $\lambda = 0, 1, 5, 10$.

INFLUENCE DE LA TAILLE DU MODÈLE. La Figure 10 et la Figure 11 montrent que les modèles plus grands produisent des textures plus détaillées, mais un modèle intermédiaire ($C = 32$) offre un compromis intéressant entre qualité et coût.

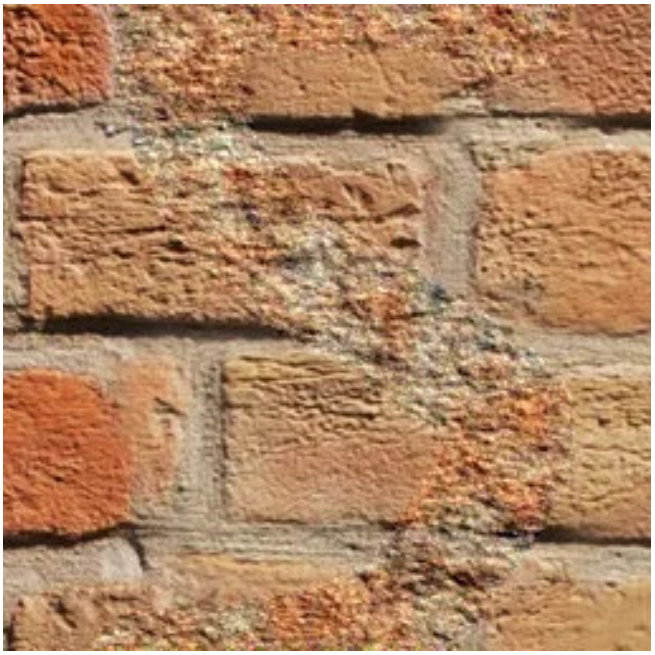


Figure 10: Petit modèle ($C = 8$).

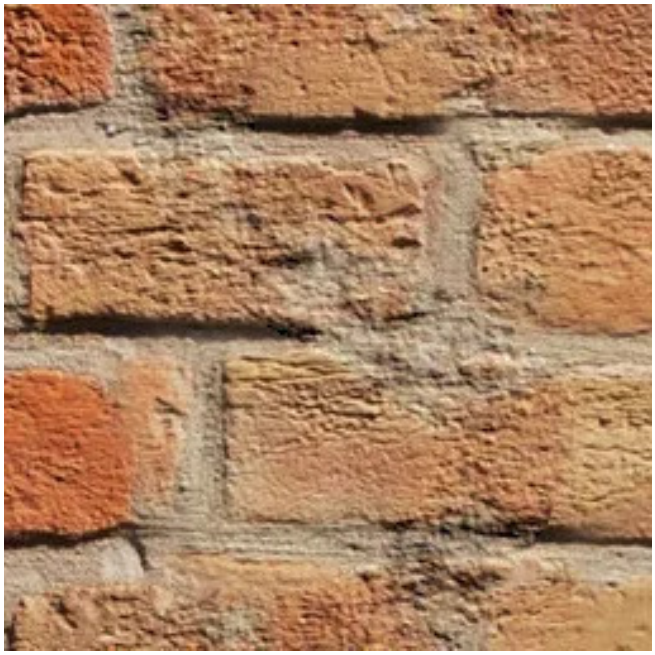


Figure 11: Modèle intermédiaire ($C = 32$).

.5.2 *Méthode B : Inpainting par Transport direct*

La seconde méthode produit des complétions plus cohérentes, avec des transitions plus douces dans certaines textures, et une meilleure qualité visuelle.

INFLUENCE DE LA CAPACITÉ DU MODÈLE. Comme montré Figure 13, un modèle de taille intermédiaire ($C = 32$) est suffisant pour capturer les structures globales tout en limitant le coût mémoire.

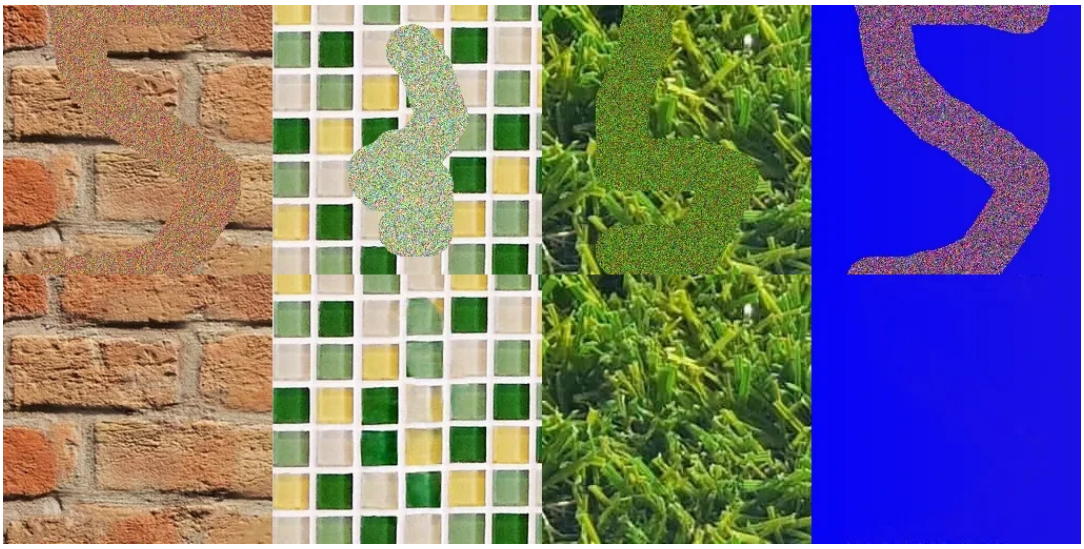


Figure 12: Résultats qualitatifs de la seconde méthode. Ligne du haut : image dégradée. Ligne du bas : complétion générée. Inférence avec 100 pas d'Euler.

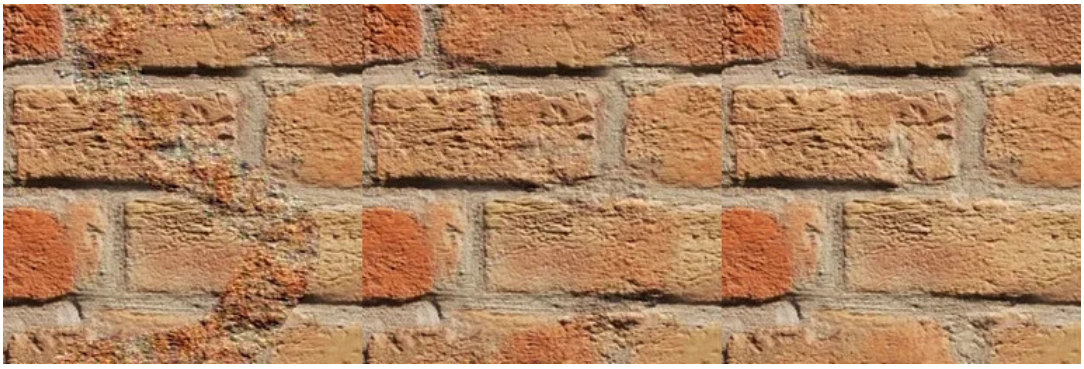


Figure 13: Effet de la taille du modèle. De gauche à droite : $C = 8, 32, 128$.

.6 DISCUSSION

Les deux méthodes d'inpainting explorées reposent sur le *Flow Matching* mais se distinguent par leur manière de gérer le conditionnement et la fidélité aux données observées, ainsi que par leur compromis fidélité/diversité.

MÉTHODE A : CLASSIFIER-FREE GUIDANCE. Cette approche offre un contrôle explicite du compromis fidélité/diversité via le paramètre λ. Pour $\lambda \approx 1$, la complétion tend vers un échantillon de la distribution conditionnelle $p(X_1 \mid \tilde{X})$, assurant une fidélité raisonnable aux parties observées. L'augmentation de λ accentue la guidance, ce qui peut améliorer la cohérence globale mais introduit des artefacts aux bords du masque. Localement, le modèle conserve une bonne cohérence des textures et des détails, mais la structure globale peut être affectée si le paramètre est mal choisi. Cette méthode nécessite deux évaluations du modèle par pas de temps, ce qui double le coût de calcul par rapport à la méthode B.

MÉTHODE B : TRANSPORT DIRECT. Basée sur le champ de vitesse qui interpole entre l'image dégradée et l'image complète, cette méthode favorise la continuité structurelle et la fidélité visuelle. La réinjection des pixels observés empêche toute altération des zones connues. Bien que cette approche ne permette pas un contrôle direct de la diversité et ne garantisse pas théoriquement un échantillonnage exact de la distribution conditionnelle, elle présente une fidélité structurelle élevée, renforcée par les principes développés dans *Rectified Flow* [59]. Contrairement à la méthode A, elle est plus efficace en termes de calcul, ne nécessitant qu'une seule évaluation par pas.

EFFICACITÉ ET CONVERGENCE. Pour les deux méthodes, une intégration explicite d'Euler avec 20 à 100 pas suffit à produire des complétions de qualité. Cette rapidité est directement liée à la nature déterministe du *Flow Matching* et à l'encodage approprié des masques dans le modèle. Les complétions obtenues restent cohérentes même avec un nombre réduit de pas, ce qui souligne l'efficacité de la formulation probabiliste.

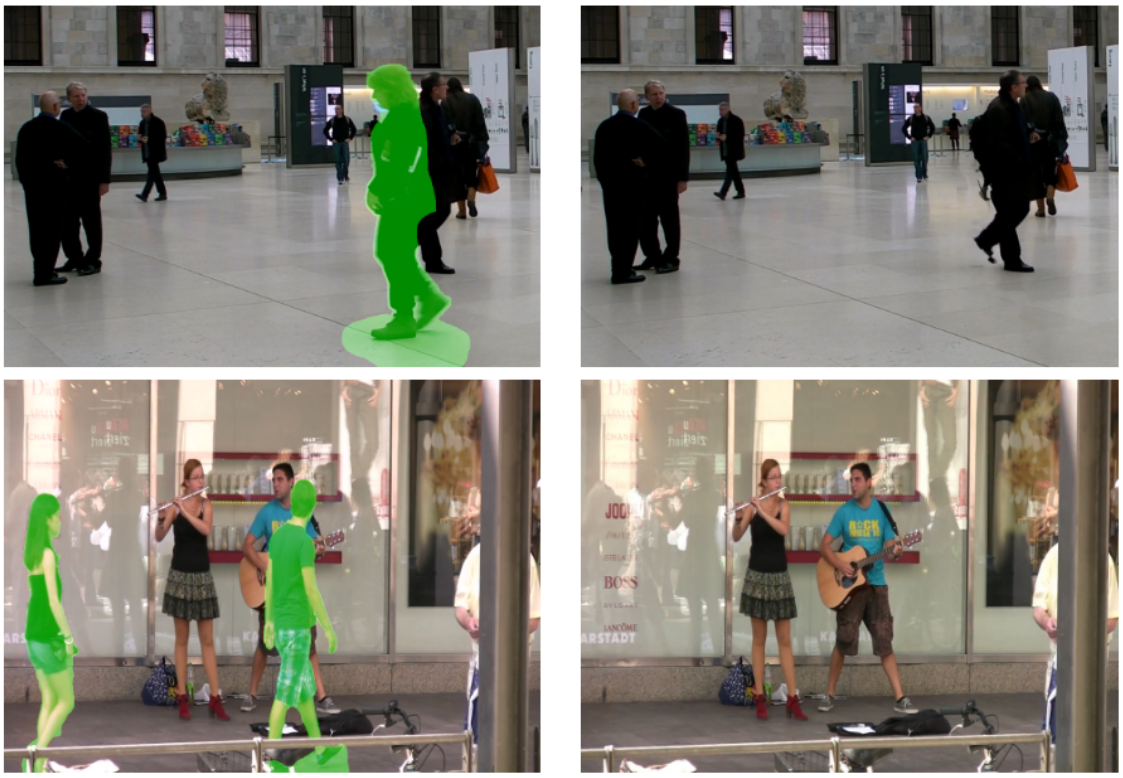
LIMITES ET PERSPECTIVES. Les méthodes présentées sont limitées à l'image test et n'offrent pas de généralisation automatique à d'autres images ou contextes. La méthode A peut générer des artefacts locaux si le paramètre λ est trop élevé, tandis que la méthode B, bien qu'assurant une forte fidélité structurelle, ne propose aucun mécanisme explicite de régulation de la diversité. Des travaux futurs pourraient explorer l'extension à un apprentissage multi-image pour améliorer la généralisation.

.7 CONCLUSION

Ce chapitre a formalisé l'inpainting comme un problème de génération conditionnelle et a montré que le *Flow Matching* permet de produire des complétions cohérentes avec un nombre faible de pas d'intégration. Deux approches ont été étudiées : (i) la classifier-free guidance, offrant un contrôle sur le compromis fidélité/diversité, et (ii) le transport optimal, qui favorise la fidélité aux zones observées grâce à la réinjection explicite des parties connues. Les résultats suggèrent que le transport direct fournit une qualité visuelle plus élevée et a été retenu pour l'application à l'inpainting vidéo, objet du chapitre suivant.

Partie IV

INPAINTING DE VIDÉOS



1

1 Illustration tirée de [69]

INPAINTING VIDÉO

Dans les chapitres précédents, nous avons développé des méthodes d'inpainting d'images basées sur le Flow Matching, exploitant l'auto-similarité des images pour produire des complétions de bonne qualité avec des architectures légères. Nous avons également démontré qu'il est possible d'obtenir des résultats visuellement satisfaisants sans recourir à de larges modèles pré-entraînés sur d'importantes bases de données. Nous nous tournons à présent vers l'inpainting vidéo, qui constitue une extension naturelle mais non triviale de nos travaux précédents. L'ajout de la dimension temporelle introduit de nouveaux défis : maintenir la cohérence temporelle entre les images successives, gérer efficacement les mouvements de caméra et d'objets, tout en préservant la frugalité computationnelle qui a guidé notre approche jusqu'ici. Ces contraintes sont d'autant plus critiques que le coût d'inférence des modèles de diffusion croît linéairement avec le nombre d'images à traiter, rendant l'inpainting vidéo particulièrement gourmand en ressources. Dans ce chapitre, nous proposons une extension directe de notre seconde méthode d'inpainting d'images au cas vidéo. Notre objectif est double : démontrer qu'il est possible de réaliser de l'inpainting vidéo avec des modèles de moins de 500 000 paramètres, tout en maintenant une cohérence temporelle satisfaisante. Nous commençons par un état de l'art des principales approches d'inpainting vidéo, avant de présenter notre méthode et nos résultats expérimentaux.

.1 ÉTAT DE L'ART

Cet état de l'art se base principalement sur la revue de littérature de Cherel et al. (2024) [16] et de Quan et al. (2024) [78]. Nous allons passer en revue les principales approches d'inpainting vidéo, en les classant en quatre catégories : les méthodes classiques, les méthodes basées sur le flot optique, les méthodes par apprentissage profond, et les méthodes attentionnelles.

.1.1 *Méthodes classiques*

Les premières méthodes reposaient sur l'exploitation d'informations déjà présentes dans la séquence vidéo afin de remplir les régions masquées, en prolongeant les approches par patchs d'images (Criminisi et al., 2004 [19] ; Drori et al., 2003 [24]). Les travaux initiaux distinguaient deux cas : celui, plus simple, d'un arrière-plan statique masqué par un objet en mouvement, et celui, plus complexe, d'un objet mobile lui-même occulté (Patwardhan et al., 2005 [74]). Pour gérer de légers mouvements de caméra, une étape de prétraitement a été ajoutée par la suite (Patwardhan et al., 2007 [75]).

Les approches gloutonnes par patchs ont ensuite été remplacées par une formulation en optimisation globale (Wexler et al., 2007 [107]), permettant une meilleure cohérence spatio-temporelle, mais au prix de temps de calcul élevés. Newson et al. (2014) [70] ont amélioré cette stratégie en intégrant des descripteurs de texture, une étape de stabilisation et une adaptation de l'algorithme PatchMatch (Barnes et al., 2009) [3] , accélérant nettement la reconstruction. Bien que coûteuses, les méthodes par patchs peuvent propager l'inpainting aux nouvelles images en exploitant les mouvements de caméra (Herling et Broll, 2014 [34]). Le et al. (2017) [49] enrichissent encore la description des patchs par le flot optique et les déforment en fonction de ce dernier pour mieux traiter les objets en mouvement. Granados et al. (2012) [30] optimisent un shift-map capable de gérer des mouvements non périodiques, mais nécessitant la résolution de problèmes de graph cut très coûteux.

.1.2 *Méthodes basées sur le flot optique*

Une autre famille d'approches exploite directement le flot optique, particulièrement utile lorsque la région masquée est visible à un autre moment de la vidéo. Shiratori et al. (2006) [88] ont montré qu'il est souvent plus simple de compléter le flot que les valeurs de couleur elles-mêmes. Matsushita et al. (2006) [65] utilisent cette idée pour stabiliser des vidéos en plein cadre. Strobel et al. (2014) [94] réalisent d'abord l'inpainting du flot, ce qui ajoute des contraintes à l'étape de restauration des couleurs par patchs. Huang et al. (2016) [40] optimisent conjointement l'apparence et le flot pour assurer la cohérence temporelle. Bokov et Vatolin (2018) [7] proposent une optimisation jointe encore plus rapide, capable de gérer tout type de mouvement de caméra. Xie et al. (2017) [110] appliquent ce principe à la synthèse de textures dynamiques.

.1.3 *L'essor des méthodes par apprentissage profond*

Avec l'explosion des données disponibles, les méthodes d'apprentissage profond se sont multipliées. Si elles surpassent souvent les approches traditionnelles en termes de qualité, elles restent limitées par des contraintes mémoire importantes, ce qui rend l'inpainting en haute résolution encore difficile. Kim et al. (2019) [43] proposent un réseau 3D-2D avec une perte de cohérence de flot optique. Wang et al. (2019) [105] décomposent la tâche en deux étapes : une inpainting à basse résolution par convolutions 3D, suivi d'une restauration par image. Chang et al. (2019) [14] étendent en 3D les *gated convolutions* [114] via un Patch-GAN 3D.

Murase et al. (2019) [67] développent un réseau léger pour l'inpainting du flot, ensuite intégré à la reconstruction. Gao et al. (2020) [27] et Xu et al. (2019) [111] suivent une approche similaire : compléter d'abord le flot optique puis propager les pixels. Ces méthodes gèrent efficacement les déformations non rigides et les dépendances à longue portée. Zhang et al. (2022c) [116] améliorent la précision du flot en exploitant plus d'images. Li et al. (2022b) [52] intègrent la prédiction et la propagation du flot directement dans la boucle d'apprentissage afin d'optimiser le réseau pour l'inpainting vidéo.

.1.4 *Méthodes attentionnelles et variantes*

Les algorithmes basés sur l'attention peuvent être vus comme des évolutions des approches par patchs, mais restent contraints par les limitations mémoire liées à la résolution vidéo. Oh et al. (2019) [73] revisitent une stratégie gloutonne par « pelage d'oignon ». Lee et al. (2019) [50] améliorent les méthodes de copie-collage avec l'apprentissage profond et des transformations affines pour aligner les images. Zeng et al. (2020) [115] alignent les images et utilisent un Transformer pour combler plusieurs régions simultanément. Hu et al. (2020) [39] combinent différentes propositions dépassant le niveau pixel. Zhang et al. (2022b) [116] couplent Transformer et flot optique. Plus récemment, Propainter (Zhou et al., 2023) [122] améliore la propagation guidée par flot et réduit les calculs inutiles des modules attentionnels en restreignant l'opération aux seules requêtes masquées.

.2 MÉTHODE

Notre approche est une extension directe de la **Méthode B** (flot et transport direct) développée pour l'inpainting d'images au cas vidéo. Similairement au chapitre précédent où nous entraînions le modèle directement sur l'image de test via *internal learning*, nous entraînons ici notre modèle sur une unique vidéo, découpée en fenêtre temporelles. Cependant, dû à la nature très différente des vidéos par rapport aux grandes images de textures, nous entraînons le modèle directement sur les fenêtres temporelles de la vidéo plutôt que sur des tranches spatiales en plus de temporelles.

.2.1 *Protocole expérimental*

.2.1.1 *Principe général*

Notre méthode repose sur l'apprentissage d'un modèle de Flow Matching capable de restaurer des fenêtres vidéo dégradées. Le modèle est entraîné à prédire un champ de vitesses transportant les fenêtres dégradées vers leur version complète, en exploitant la cohérence spatio-temporelle de la vidéo.

À la différence de l'inpainting d'images où nous exploitions uniquement l'auto-similarité d'une seule image via *internal learning*, nous nous appuyons ici sur un ensemble de fenêtres vidéo d'entraînement. Cette stratégie permet au modèle d'apprendre des motifs de mouvement et de texture récurrents dans les vidéos, tout en maintenant une architecture légère (moins de 500 000 paramètres).

.2.1.2 *Représentation des données vidéo*

Une vidéo dégradée est notée $\tilde{X} \in \mathbb{R}^{T\times C\times H\times W}$, avec :

- T : nombre total d'images dans la vidéo,
- C : nombre de canaux (typiquement 3 pour RGB),
- $H \times W$: résolution spatiale de chaque image.

Le masque binaire associé, définissant les régions à restaurer, est noté $M \in \{0,1\}^{T\times 1\times H\times W}$. Une valeur de 1 indique un pixel à compléter, tandis qu'une valeur de 0 correspond à un pixel observé.

Fenêtres temporelles. Pour des raisons de mémoire GPU (les vidéos comptant entre 80 et 200 images), le modèle ne peut pas traiter l'intégralité de la vidéo simultanément. Nous découpons donc la vidéo en **fenêtres temporelles glissantes** de longueur $N < T$ (voir Figure 14) :

$$x_{t,n} = \tilde{x}_t[n : n+N] \in \mathbb{R}^{N\times C\times H\times W},$$

où $n \in \{0,1,\dots,T-N\}$ désigne l'indice de début de la fenêtre et où $\tilde{x}_t$ est la vidéo échantillonnée au temps de diffusion t. Le masque correspondant est noté $M_n = M[n : n+N]$.

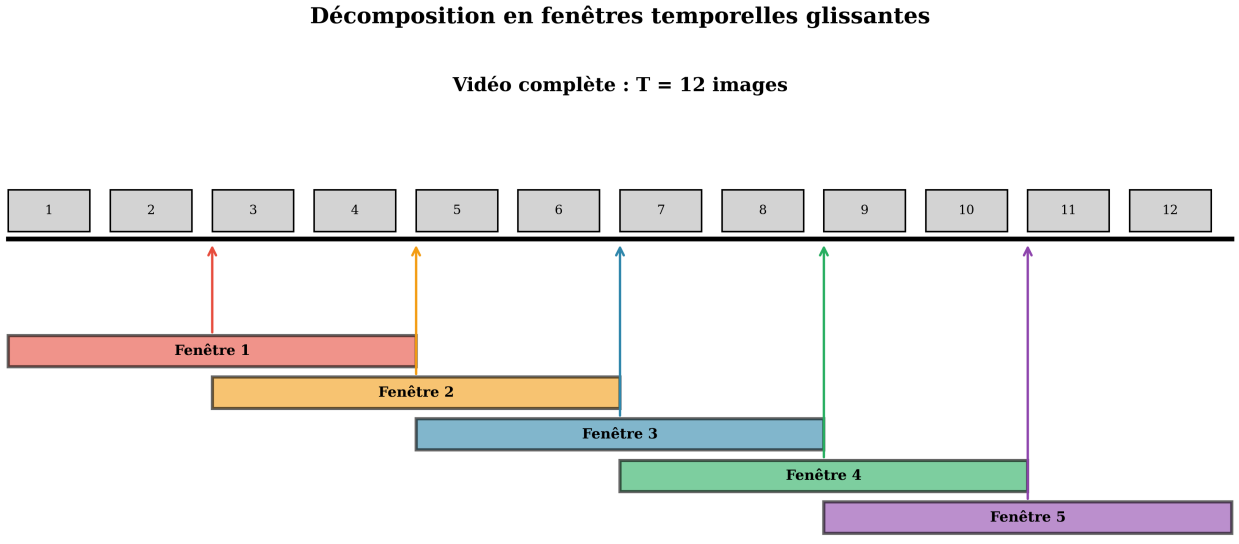


Figure 14: Décomposition d'une vidéo en fenêtres temporelles glissantes. Pour une vidéo de $T = 12$ images et des fenêtres de taille $N = 4$, nous obtenons 9 fenêtres avec chevauchement (nous n'en montrons que 5 ici). Chaque fenêtre est traitée indépendamment par le modèle, puis les prédictions sont agrégées pour reconstruire la vidéo complète.

.2.1.3 *Préparation des données d'entraînement*

La Figure 15 illustre le pipeline complet de préparation des données d'entraînement. À partir d'une fenêtre propre X_1, nous générons des masques aléatoires puis créons la fenêtre dégradée X_0 en remplaçant les régions masquées par du bruit gaussien. Le modèle apprend ensuite à interpoler linéairement entre X_0 et X_1 via le champ de vitesses.

Masques d'entraînement. À chaque itération d'entraînement, pour chaque fenêtre temporelle, nous générons aléatoirement un masque $M_{n,\text{train}} \sim p_M$ suivant la procédure de Cherel et al. [16], qui simule des coups d'effaceur aléatoires via l'algorithme de Yu et al. (2019) [114]. Ces masques peuvent être :

- **Statiques** : identiques sur toutes les images de la fenêtre, simulant l'effacement d'une région fixe (par exemple, un objet statique à supprimer),
- **Dynamiques** : variant d'une image à l'autre pour simuler des objets en mouvement ou des occlusions temporelles.

La couverture de ces masques varie typiquement entre 15% et 50% de la surface de l'image. Nous utilisons une banque de 15 000 masques pré-générés, échantillonnés uniformément durant l'entraînement.

Masques de test. L'utilisateur fournit un masque de test $M_{n,\text{test}}$ définissant les régions à restaurer dans sa vidéo. Ce masque reste fixe durant tout le processus d'inférence et d'entraînement. Comme le montre la Figure 15, les zones correspondant à $M_{n,\text{test}}$ ne sont jamais observées durant l'entraînement (elles apparaissent en noir dans les visualisations), garantissant ainsi que le modèle apprend uniquement sur les masques d'entraînement.

Masque combiné. Le masque final utilisé lors de l'entraînement combine les deux types de masques :

$$M_n = 1 - (1 - M_{n,\text{train}}) \odot (1 - M_{n,\text{test}}),$$

où $\odot$ désigne le produit élément par élément. Cette formulation garantit que les pixels masqués par $M_{n,\text{test}}$ ne sont jamais utilisés pour l'entraînement, évitant ainsi toute fuite d'information.

Prétraitement. Nous appliquons la même méthodologie de normalisation que dans le chapitre précédent. Chaque canal est normalisé indépendamment pour avoir une moyenne nulle et une variance unitaire. Pour chaque canal $c \in \{R, G, B\}$, la normalisation s'écrit :

$$\tilde{x}_c = \frac{x_c - \mu_c}{\sigma_c},$$

où les statistiques $\mu_c = \mathbb{E}[x_c]$ et $\sigma_c = \sqrt{\mathbb{V}[x_c]}$ sont calculées sur l'ensemble des pixels de la vidéo complète. Cette normalisation stabilise l'entraînement en centrant les gradients et accélère la convergence.

.2.2 *Architecture du modèle*

.2.2.1 *U-Net 3D*

Notre architecture repose sur un U-Net 3D, extension naturelle du U-Net 2D au domaine spatio-temporel. Cette architecture se compose de :

- **Encodeur** : quatre niveaux de résolution, chacun comportant deux blocs résiduels composés de :
  - Convolutions 3D avec noyaux $3 \times 3 \times 3$,
  - Normalisation par batch (BatchNorm3D),
  - Fonction d'activation ReLU.

  Entre chaque niveau, un downsampling spatial de facteur 2 est appliqué (stride 2 dans les dimensions spatiales uniquement, la dimension temporelle restant inchangée).
- **Bloc du milieu** : deux blocs résiduels à la résolution la plus basse.

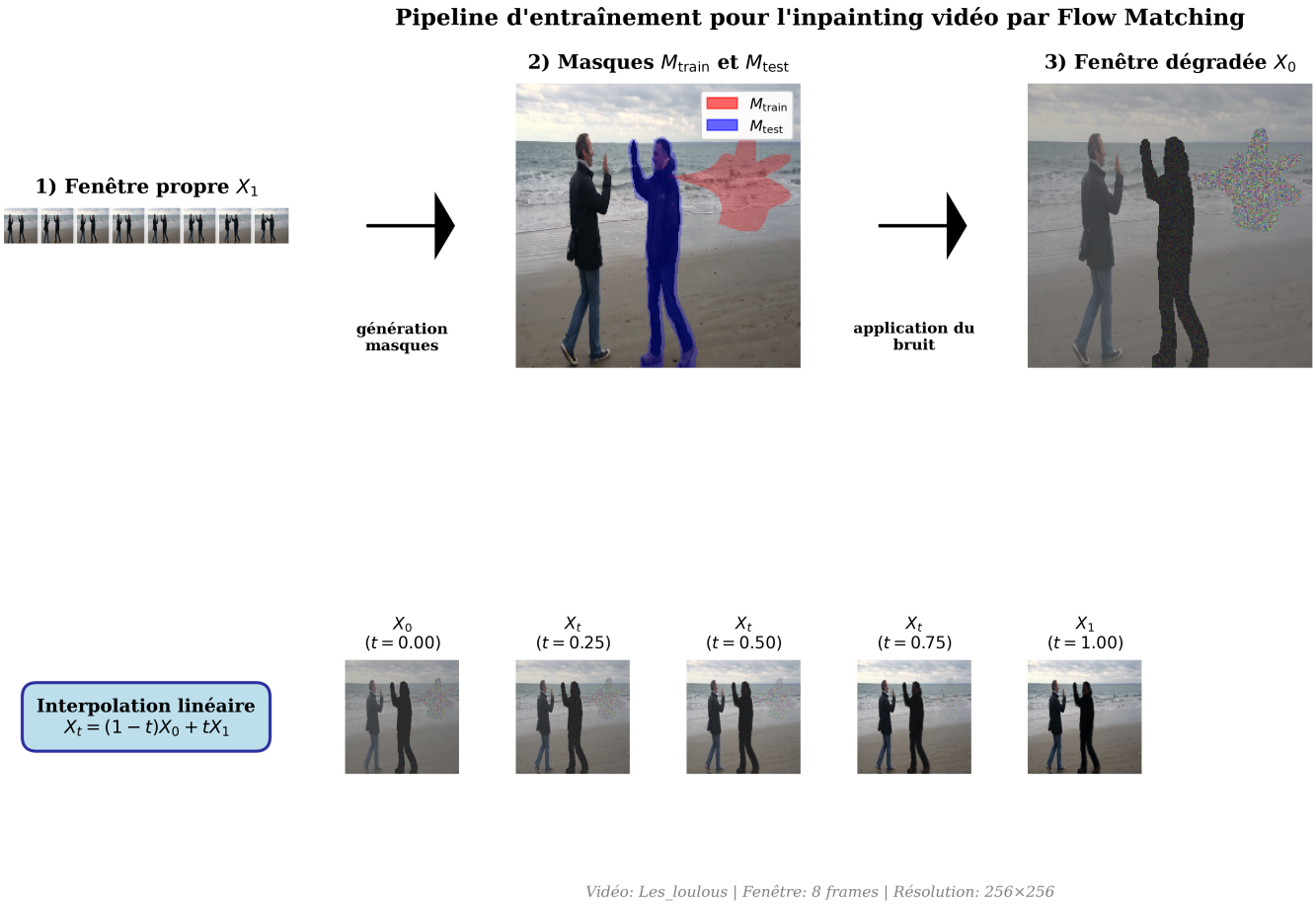


Figure 15: Pipeline d'entraînement pour l'inpainting vidéo. **(Haut)** À partir d'une fenêtre propre X_1, nous générons des masques M_{train} (rouge) et M_{test} (bleu), puis créons la fenêtre dégradée X_0 en remplaçant les zones masquées par du bruit gaussien. La zone M_{test} (en noir) n'est jamais visible durant l'entraînement. **(Bas)** Interpolation linéaire $X_t = (1-t)X_0 + tX_1$ montrant la transition progressive de la fenêtre dégradée ($t = 0$) vers la fenêtre propre ($t = 1$). Le modèle apprend à prédire le champ de vitesses permettant cette transition.

- **Décodeur** : symétrique à l'encodeur, avec upsampling spatial par interpolation bilinéaire et skip connections depuis les niveaux correspondants de l'encodeur.
- **Largeur des couches** : fixée à 32 canaux pour tous les niveaux, garantissant une empreinte mémoire réduite (environ 480 000 paramètres au total).

Cette architecture légère contraste fortement avec les modèles de l'état de l'art qui comptent typiquement plusieurs millions, voire centaines de millions de paramètres. Le choix d'un U-Net 3D plutôt que d'architectures plus sophistiquées (comme les Transformers spatio-temporels) découle directement de notre objectif de frugalité : les convolutions 3D offrent un bon équilibre entre capacité de modélisation de la cohérence temporelle et efficacité computationnelle. De plus, nous avons essayé d'appliquer *FastDVDNet* [97] pour notre tâche, mais ses performances étaient loin d'être aussi fructueuses qu'escompté.

.2.2.2 *Encodage temporel*

Le temps de diffusion $t \in [0, 1]$ est encodé via des **Fourier Features** [96], suivant la même approche que dans le chapitre précédent. Formellement, pour un vecteur de fréquences $\boldsymbol{\omega} \in \mathbb{R}^d$, l'encodage est :

$$\phi(t) = \begin{bmatrix} \sin(2\pi\omega_1 t) \\ \cos(2\pi\omega_1 t) \\ \vdots \\ \sin(2\pi\omega_d t) \\ \cos(2\pi\omega_d t) \end{bmatrix} \in \mathbb{R}^{2d}.$$

Ce vecteur est ensuite projeté via une couche fully-connected, puis **concaténé à l'entrée** x_t du U-Net, permettant au modèle de conditionner ses prédictions sur le temps de diffusion.

.2.2.3 *Mixture of Experts*

Pour améliorer la capacité du modèle à gérer différentes étapes du processus de diffusion, nous utilisons une **mixture of experts** composée de 20 modèles spécialisés. Chaque expert est entraîné sur un intervalle distinct de t :

$$\text{Expert}_i \text{ pour } t \in \left[\frac{i-1}{20}, \frac{i}{20}\right], \quad i \in \{1, \ldots, 20\}.$$

Lors de l'inférence, pour un temps t donné, nous sélectionnons l'expert correspondant à l'intervalle contenant t. Cette approche permet :

- Une meilleure spécialisation : chaque expert se concentre sur une phase spécifique du processus (début, milieu, ou fin de la trajectoire),
- Une réduction de la variance des prédictions par rapport à un modèle unique devant couvrir tout l'intervalle $[0, 1]$,
- Un entraînement parallélisable des différents experts, réduisant le temps total d'entraînement.

Cette stratégie est particulièrement efficace dans le cadre du Flow Matching, où la complexité de la tâche varie significativement selon t : au début ($t \approx 0$), le modèle doit gérer du bruit pur, tandis qu'à la fin ($t \approx 1$), il effectue des ajustements fins sur une image quasi-restaurée.

.2.3 *Formulation en Flow Matching*

Notre modèle suit le paradigme du **Flow Matching**, identique à la Méthode B du chapitre précédent. Nous formulons le problème comme un transport entre une distribution d'origine (fenêtres dégradées) et une distribution cible (fenêtres complètes).

.2.3.1 *Distributions et conditionnement*

- **Distribution cible :** $X_1 \sim p_{\text{data}}$, les fenêtres de longueur N extraites de la vidéo propre.
- **Distribution d'origine :**
$$X_0 = X_1 \odot (1 - M_n) + Z \odot M_n,$$
où $Z \sim \mathcal{N}(0, I)$ représente un bruit gaussien standard et M_n est le masque combiné défini précédemment. Autrement dit, X_0 conserve les pixels non masqués de X_1 et remplace les pixels masqués par du bruit.
- **Conditionnement :** le masque combiné $Y = M_n$.

- **Interpolation linéaire :** le chemin entre X_0 et X_1 est défini par
$$X_t = (1-t)X_0 + tX_1, \quad t \sim \mathcal{U}[0,1].$$
- **Entrée du modèle :** $(x_{t,n}, M_n) \in \mathbb{R}^{N\times(C+1)\times H\times W}$, obtenu par concaténation de la fenêtre bruitée et de son masque le long de la dimension des canaux.
- **Sortie du modèle :** prédiction du champ de vitesses $u^{\theta}_{t,n}(x_{t,n}, M_n) \in \mathbb{R}^{N\times C\times H\times W}$.

.2.3.2 *Fonction de perte*

La fonction de perte mesure l'écart entre le champ de vitesses théorique $(x_1 - x_0)$ et la prédiction du modèle, **uniquement sur les pixels masqués par $M_{n,\text{train}}$ et non masqués par $M_{n,\text{test}}$** :

$$\mathcal{L}(\theta) = \mathbb{E}_{t,\, x_1,\, M_{\text{train}},\, z}\left[\left\|\left((x_1 - x_0) - u^{\theta}_t(x_t, M_n)\right) \odot M_{n,\text{train}} \odot (1 - M_{n,\text{test}})\right\|_2^2\right],$$

avec

$$x_0 = x_1 \odot (1 - M_n) + z \odot M_n,$$
$$x_t = (1-t)x_0 + tx_1.$$

Cette formulation garantit que :

1. Le modèle n'apprend jamais sur la région de test $M_{n,\text{test}}$, évitant toute fuite d'information,
2. Le modèle se concentre uniquement sur la restauration des zones masquées durant l'entraînement,
3. La cohérence spatio-temporelle est préservée par les convolutions 3D qui traitent simultanément les dimensions spatiales et temporelles.

Le terme $M_{n,\text{train}} \odot (1 - M_{n,\text{test}})$ est crucial : il masque à la fois les pixels non corrompus dans les données d'entraînement (terme $M_{n,\text{train}}$) et les pixels correspondant au masque de test (terme $(1 - M_{n,\text{test}})$).

.2.3.3 *Sans guidance*

Contrairement à la Méthode A du chapitre précédent qui utilisait du *classifier-free guidance*, nous n'employons ici **aucune forme de guidance**. Le modèle reçoit uniquement $(x_{t,n}, M_n)$ en entrée et prédit directement le champ de vitesses. Cette simplification s'est révélée suffisante pour obtenir des résultats de bonne qualité tout en réduisant le coût computationnel :

- Pas besoin de double évaluation du modèle (avec et sans conditionnement),
- Inférence deux fois plus rapide,

.2.4 *Fenêtres temporelles et inférence*

.2.4.1 *Découpage en fenêtres*

Pour une vidéo de longueur T et des fenêtres de taille N, nous obtenons $T - N + 1$ fenêtres glissantes avec un overlap de $N - 1$ images entre fenêtres consécutives (voir Figure 14) :

$$\text{Fenêtre}_n = x[n : n + N], \quad n \in \{0, 1, \ldots, T - N\}.$$

Chaque fenêtre est traitée indépendamment par le modèle, produisant une prédiction de vélocité $u^{\theta}_{t,n}(x_{t,n}, M_n)$ de même dimension. Ce découpage présente plusieurs avantages :

- **Mémoire GPU maîtrisée** : traiter N images simultanément plutôt que T permet de gérer des vidéos arbitrairement longues,
- **Cohérence locale** : les convolutions 3D capturent les dépendances temporelles au sein de chaque fenêtre,
- **Parallélisation** : plusieurs fenêtres peuvent être traitées en parallèle (par lots).

.2.4.2 *Agrégation des prédictions*

Lors de l'inférence, les $T - N + 1$ prédictions doivent être agrégées pour reconstruire la vélocité de la vidéo complète. Nous avons exploré deux stratégies principales (voir Figure 16) :

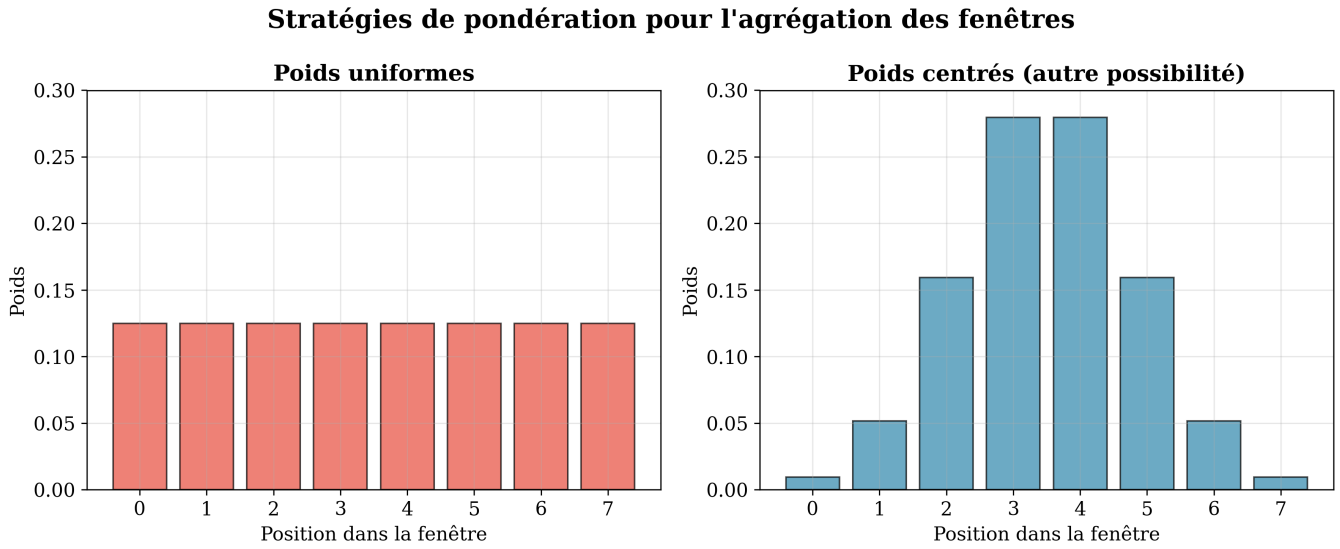


Figure 16: Stratégies de pondération pour l'agrégation des fenêtres. **(Gauche)** Poids uniformes : toutes les positions dans une fenêtre reçoivent le même poids, conduisant à une moyenne simple. **(Droite)** Poids centrés : les images centrales de chaque fenêtre reçoivent un poids supérieur, réduisant l'influence des prédictions aux bords qui sont généralement moins fiables.

Stratégie actuelle : moyenne uniforme. Nous utilisons actuellement une moyenne glissante simple :

$$u_t^\theta(x_t, M)[i] = \frac{1}{|\mathcal{W}_i|} \sum_{n \in \mathcal{W}_i} u_{t,n}^\theta(x_{t,n}, M_n)[i-n],$$

où $\mathcal{W}_i = \{n : 0 \leqslant i - n < N\}$ désigne l'ensemble des fenêtres contenant l'image i. Cette approche traite toutes les positions d'une fenêtre de manière égale.

Stratégie envisagée : pondération centrée. Comme illustré dans la Figure 16, nous pensons qu'**une pondération privilégiant les images centrales** de chaque fenêtre améliorerait significativement les résultats. Une telle pondération pourrait suivre une distribution gaussienne centrée sur $N/2$:

$$w_k = \exp\left(-\frac{(k - N/2)^2}{2\sigma^2}\right), \quad k \in \{0, \ldots, N-1\},$$

donnant une agrégation pondérée :

$$u_t^\theta(x_t, M)[i] = \frac{\sum_{n \in \mathcal{W}_i} w_{i-n} \cdot u_{t,n}^\theta(x_{t,n}, M_n)[i-n]}{\sum_{n \in \mathcal{W}_i} w_{i-n}}.$$

Cette amélioration reste à implémenter et évaluer. L'intuition sous-jacente est que les prédictions aux bords des fenêtres sont moins fiables car elles disposent de moins de contexte temporel, tandis que les images centrales bénéficient pleinement du champ réceptif temporel du modèle.

.2.4.3 *Optimisation de l'inférence*

Traitement par lots. Pour maximiser l'utilisation du GPU, nous traitons les fenêtres par lots de taille B. Pour un temps t fixé, nous calculons simultanément les prédictions pour B fenêtres consécutives :

$$\{u_{t,n}^\theta, u_{t,n+1}^\theta, \ldots, u_{t,n+B-1}^\theta\} = \text{Modèle}\big(\{x_{t,n}, x_{t,n+1}, \ldots, x_{t,n+B-1}\}\big).$$

Équilibrage mémoire-vitesse. La taille de lot B est ajustée dynamiquement en fonction de la mémoire GPU disponible et de la longueur N des fenêtres. Typiquement, pour $N = 8$ et une résolution de $256 \times$

256, nous utilisons $B = 16$ sur une NVIDIA A100, ce qui permet de traiter 128 images simultanément ($B \times N = 16 \times 8$).

Stabilisation par réinjection du contexte. Similairement à la Méthode B du chapitre précédent et à l'astuce de RePaint [64], nous réinjectons à chaque pas d'intégration les pixels observés :

$$\tilde{u}_t^\theta(x_t, M) = u_t^\theta(x_t, M) \odot M + (x_1 - x_0) \odot (1 - M),$$

où $(x_1 - x_0)$ représente le champ de vitesses conditionnel théorique pour le scheduler linéaire. Cette correction force le modèle à ne modifier que les régions masquées, stabilisant la reconstruction et empêchant toute dérive dans les zones observées. En pratique, cette opération améliore nettement la cohérence visuelle aux frontières du masque.

.2.5 *Hyperparamètres d'entraînement*

Le tableau 4 résume les principaux hyperparamètres utilisés :

Paramètre	**Valeur**
Longueur des fenêtres (N)	8 images
Résolution spatiale	256×256
Nombre de canaux	32
Nombre de paramètres	~480 000
Nombre d'experts	20
Optimiseur	AdamW [62]
Taux d'apprentissage	3×10^{-4}
Batch size	64 fenêtres
Nombre d'itérations	10 000 par expert
Échantillonnage de t	LogitNormal($\mu = 0, s = 1$)
Solveur (inférence)	Euler explicite
Nombre de pas (inférence)	100
Stratégie d'agrégation	Moyenne uniforme
Durée d'entraînement (1 expert)	~2h (A100 40Go)

Table 4: Hyperparamètres d'entraînement et d'inférence pour l'inpainting vidéo

Justification des choix :

- $N = 8$: compromis entre cohérence temporelle (contexte suffisant) et mémoire GPU. Des fenêtres plus longues amélioreraient la modélisation des mouvements lents mais nécessiteraient plus de mémoire.
- **32 canaux** : limite volontaire pour rester sous 500 000 paramètres, conformément à notre objectif de frugalité.
- **20 experts** : division fine de l'intervalle $[0, 1]$ pour améliorer la précision sans trop complexifier l'entraînement. Chaque expert couvre 5% de l'intervalle temporel.
- **LogitNormal** : distribution recommandée par Stable Diffusion 3 [26] pour un échantillonnage plus efficace de t, concentrant les exemples d'entraînement sur les régions les plus critiques.
- **100 pas Euler** : bon compromis qualité/vitesse, générant une vidéo en quelques minutes sur GPU grand public. Des méthodes d'intégration plus sophistiquées (Runge-Kutta) pourraient réduire ce nombre.

- **Scheduler linéaire** : conformément aux recommandations de [26], le scheduler linéaire offre les meilleures performances pour notre tâche de transport optimal.

L'entraînement d'un expert prend environ 2 heures sur une NVIDIA A100 40Go. L'entraînement des 20 experts peut être entièrement parallélisé, ramenant le temps total à environ 2–3 heures si l'on dispose de 20 GPUs, ou à environ 40 heures sur un seul GPU si l'on entraîne séquentiellement.

.3 RÉSULTATS

Nous présentons dans cette section les résultats de notre méthode d'inpainting vidéo sur plusieurs vidéos de test. L'ensemble des résultats vidéo est disponible à cette adresse. Nous analysons qualitativement les performances de notre approche, comparons l'impact de la longueur des fenêtres temporelles, et discutons des limitations observées.

.3.1 *Protocole d'évaluation*

.3.1.1 *Vidéos de test*

Nous avons évalué notre méthode sur plusieurs vidéos de test présentant des caractéristiques variées :

- **Durée** : entre 80 et 200 images (~3–7 secondes à 30 fps),
- **Résolution** : 256×256 pixels,
- **Contenu** : scènes naturelles avec mouvements de caméra et d'objets, différents niveaux de complexité de texture.

.3.1.2 *Configurations testées*

Nous avons évalué deux configurations principales :

- $N = 8$: fenêtres de 8 images (~0.27 secondes),
- $N = 16$: fenêtres de 16 images (~0.53 secondes).

Chaque configuration utilise 20 experts entraînés pendant 10 000 itérations, avec 100 pas d'intégration Euler lors de l'inférence. Le temps d'inférence pour une vidéo de 100 images est d'environ 2.5 heures sur une NVIDIA A100.

.3.2 *Résultats qualitatifs*

.3.2.1 *Exemples de restauration*

La Figure 18 présente des exemples représentatifs de restauration vidéo pour différentes scènes. Notre méthode parvient à générer des complétions visuellement cohérentes qui s'intègrent naturellement au reste de l'image.

Points forts observés :

- **Cohérence spatiale** : les textures générées respectent généralement les structures environnantes et se fondent bien dans le reste de l'image,
- **Stabilité temporelle** : les zones restaurées présentent une bonne continuité entre images consécutives, sans scintillement majeur,
- **Préservation du mouvement** : les mouvements de caméra et d'objets sont globalement préservés dans les régions complétées,
- **Qualité des détails** : malgré l'architecture légère (480k paramètres), les détails fins sont souvent bien reconstruits.

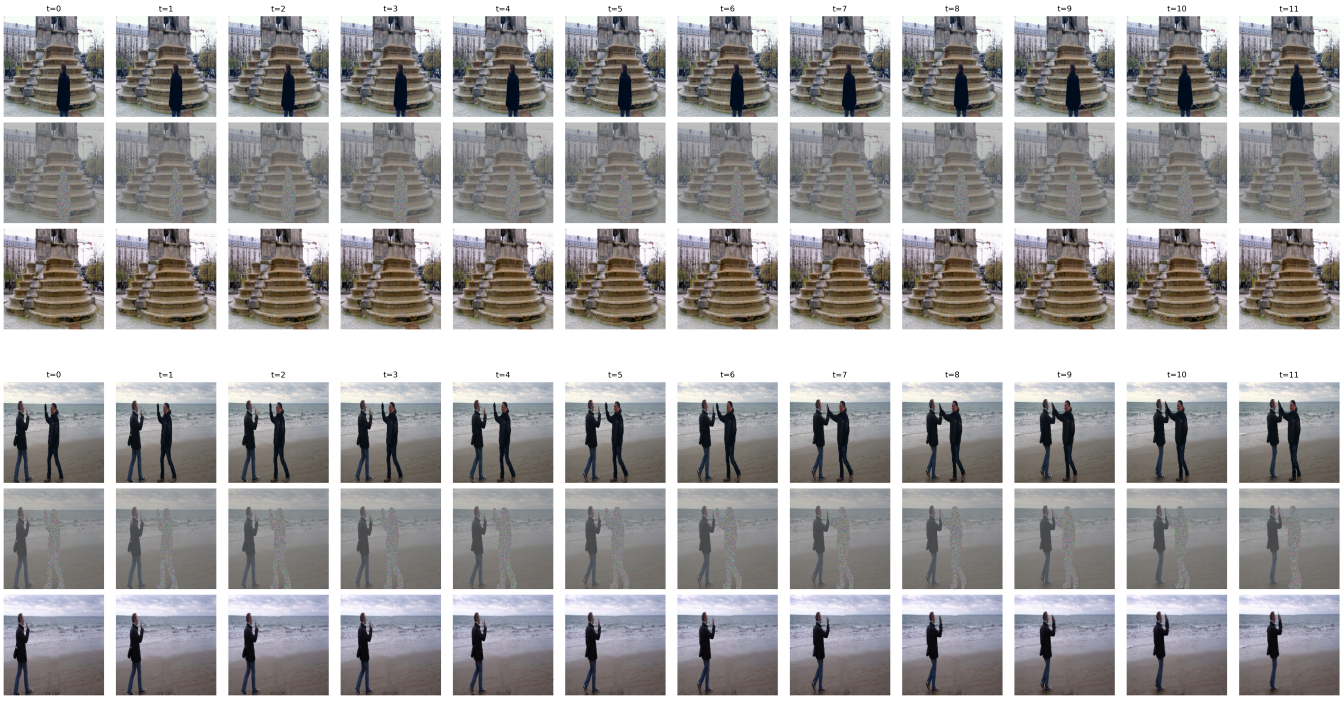


Figure 17: Exemples de restauration vidéo. La cohérence spatiale et la qualité de texture sont généralement bonnes.

.3.2.2 *Cohérence temporelle*

La Figure 17 illustre la cohérence temporelle de nos restaurations sur une séquence de 8 images consécutives. On observe une transition fluide entre les images, sans discontinuités visuelles majeures.

.3.3 *Comparaison* $N = 8$ *vs* $N = 16$

La Figure 19 compare les résultats obtenus avec des fenêtres de longueur $N = 8$ et $N = 16$ sur la même vidéo.

Observations :

- **Qualité visuelle** : les deux configurations produisent des résultats de qualité comparable pour les scènes avec mouvements modérés,
- **Mouvements rapides** : $N = 16$ capture légèrement mieux les mouvements rapides grâce à un contexte temporel plus large,
- **Temps de calcul** : $N = 16$ est environ 1.5× plus lent que $N = 8$ en raison de fenêtres plus grandes,
- **Mémoire GPU** : $N = 16$ nécessite environ 2× plus de mémoire, limitant la taille de batch,
- **Compromis** : $N = 8$ offre le meilleur équilibre qualité/vitesse/mémoire pour notre cas d'usage.

.3.4 *Limitations et artéfacts*

Malgré des résultats encourageants, plusieurs limitations subsistent :

.3.5 *Analyse comparative informelle*

Bien que nous n'ayons pas réalisé de comparaison quantitative formelle avec l'état de l'art, nous avons effectué des observations qualitatives en visionnant les résultats de méthodes récentes (ProPainter [122], STTN [115]) sur des vidéos similaires.

Forces de notre approche :

- **Frugalité** : notre modèle (480k paramètres) est 50–100× plus léger que les méthodes de l'état de l'art (typiquement 20–50M paramètres),
- **Rapidité d'entraînement** : 2h par expert vs plusieurs jours pour les grandes architectures,
- **Simplicité** : pas de pré-entraînement nécessaire, architecture simple à implémenter et comprendre,
- **Résultats acceptables** : qualité visuelle suffisante pour de nombreux cas d'usage, malgré l'architecture minimaliste.

Faiblesses par rapport à l'état de l'art :

- **Cohérence à longue portée** : les Transformers gèrent mieux les dépendances temporelles sur plusieurs secondes,
- **Cas difficiles** : les méthodes de l'état de l'art sont plus robustes sur les scènes complexes.
- **Information indisponible** : nos modèles n'étant entraînés que sur une vidéo, si l'arrière-plan derrière les zones à remplir n'a jamais été observé, nos modèles vont très mal performer.

.3.6 *Pistes d'amélioration*

Plusieurs axes d'amélioration ont été identifiés pour des travaux futurs :

.3.6.1 *Agrégation pondérée*

Comme discuté dans une section précédente, l'implémentation d'une agrégation pondérée privilégiant les images centrales de chaque fenêtre pourrait significativement réduire les artéfacts aux bords. La pondération gaussienne proposée serait:

$$w_k = \exp\left(-\frac{(k - N/2)^2}{2\sigma^2}\right)$$

.3.6.2 *Attention linéaire*

L'intégration de mécanismes d'attention à complexité linéaire (par exemple, Linear Attention **[katharopoulos2020t**
ou Performers **[choromanski2020rethinking]**) pourrait améliorer la capture des dépendances à longue portée sans compromettre l'efficacité computationnelle. Nous avions initialement prévu d'explorer cette direction, mais le temps a manqué pour mener ces expériences à bien.

.3.6.3 *Entraînement multi-vidéo*

Il serait également envisageable d'entraîner des modèles plus grands, sur beaucoup de vidéos, en pouvant raisonnablement espérer produire des résultats corrects sur des vidéos jamais observées, sans besoin de réentraînement.

.3.7 *Synthèse*

Notre méthode d'inpainting vidéo basée sur le Flow Matching démontre qu'il est possible d'obtenir des résultats visuellement satisfaisants avec une architecture extrêmement légère (moins de 500 000 paramètres). Les principales contributions sont :

- Une formulation élégante du problème via le Flow Matching sans guidance,
- Une architecture 3D simple mais efficace pour capturer la cohérence spatio-temporelle,
- Un système de mixture of experts permettant une meilleure spécialisation,

- Des résultats encourageants démontrant la viabilité de l'approche frugale.

Les limitations identifiées (artéfacts aux bords, difficultés avec les structures complexes) constituent des axes de recherche prometteurs pour des travaux futurs, notamment via l'amélioration de l'agrégation et l'intégration d'attention efficace.

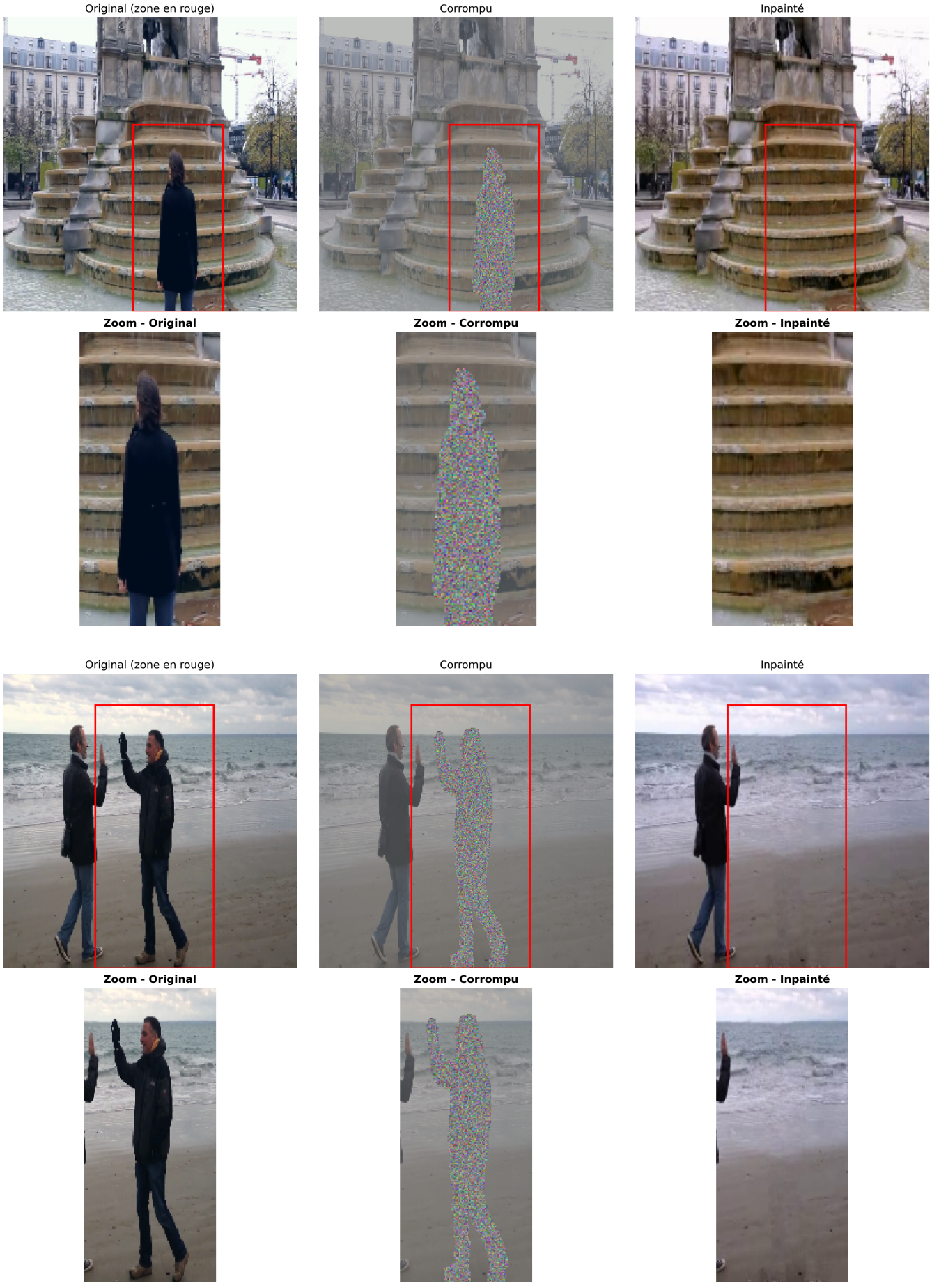


Figure 18: Zoom sur la zone à remplir.

Figure 19: Exemples de restauration vidéo. Haut: N=16. Bas: N=8.

Partie V

CONCLUSION

CONCLUSION

Tout au long de ce rapport, nous avons développé des méthodes d'inpainting basées sur des modèles par *Flow Matching*. Dans une première partie, nous avons introduit le cadre de *Flow Matching* et développé la librairie qui a servi, dans un second temps, à explorer deux méthodologies possibles pour l'inpainting d'images. De ces méthodologies, nous avons retenu celle qui semblait la plus prometteuse, et nous l'avons étendue à l'inpainting vidéo. Nous avons ainsi pu obtenir des résultats visuellement satisfaisants, tout en utilisant un modèle relativement léger (2M paramètres), entraîné sur une seule vidéo. Les prochains travaux pourraient se concentrer sur la mise à l'échelle de cette méthode, en utilisant un jeu de données constitué de bien plus qu'une seule vidéo comme nous l'avons fait. Cela permettrait d'améliorer la sémantique des complétions et éviterait de devoir réentraîner un modèle pour chaque vidéo. Il serait ensuite possible de rectifier et potentiellement distiller un tel modèle afin d'obtenir un modèle frugal d'inpainting vidéo, utilisant bien moins de pas d'Euler. De plus, notre modèle utilise actuellement un CNN, qui est connu pour son fort biais de localité. L'intégration de méthodes d'attention efficaces permettrait de modéliser des dépendances à plus long terme, et ainsi d'améliorer la cohérence temporelle et spatiale des complétions, et constitue ainsi une piste de recherche prometteuse.

Partie VI

APPENDIX

BIBLIOGRAPHIE

[1] Michael S. Albergo, Nicholas M. Boffi, and Eric Vanden-Eijnden. *Stochastic Interpolants: A Unifying Framework for Flows and Diffusions*. 2023. arXiv: 2303.08797 [cs.LG]. URL: https://arxiv.org/abs/2303.08797.

[2] Michael S. Albergo and Eric Vanden-Eijnden. *Building Normalizing Flows with Stochastic Interpolants*. 2023. arXiv: 2209.15571 [cs.LG]. URL: https://arxiv.org/abs/2209.15571.

[3] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. "PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing." In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 28.3 (Aug. 2009).

[4] Heli Ben-Hamu, Samuel Cohen, Joey Bose, Brandon Amos, Aditya Grover, Maximilian Nickel, Ricky T. Q. Chen, and Yaron Lipman. *Matching Normalizing Flows and Probability Paths on Manifolds*. 2022. arXiv: 2207.04711 [stat.ML]. URL: https://arxiv.org/abs/2207.04711.

[5] Jean-David Benamou and Yann Brenier. "A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem." In: *Numerische Mathematik* 84 (Jan. 2000), pp. 375–393. DOI: 10.1007/s002110050002.

[6] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. "Image inpainting." In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. USA: ACM Press/Addison-Wesley Publishing Co., 2000, 417–424. ISBN: 1581132085. DOI: 10.1145/344779.344972. URL: https://doi.org/10.1145/344779.344972.

[7] Alexander Bokov and Dmitriy Vatolin. "100+ Times Faster Video Completion by Optical-Flow-Guided Variational Refinement." In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. 2018, pp. 2122–2126. DOI: 10.1109/ICIP.2018.8451683.

[8] Valentin De Bortoli, Iryna Korshunova, Andriy Mnih, and Arnaud Doucet. *Schrödinger Bridge Flow for Unpaired Data Translation*. 2024. arXiv: 2409.09347 [cs.LG]. URL: https://arxiv.org/abs/2409.09347.

[9] Valentin De Bortoli, James Thornton, Jeremy Heng, and Arnaud Doucet. *Diffusion Schrödinger Bridge with Applications to Score-Based Generative Modeling*. 2023. arXiv: 2106.01357 [stat.ML]. URL: https://arxiv.org/abs/2106.01357.

[10] Yann Brenier. "Polar Factorization and Monotone Rearrangement of Vector-Valued Functions." In: *Communications on Pure and Applied Mathematics* 44 (1991), pp. 375–417. URL: https://api.semanticscholar.org/CorpusID:123428953.

[11] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. "Non-Local Means Denoising." In: *Image Processing On Line* 1 (2011). https://doi.org/10.5201/ipol.2011.bcm_nlm, pp. 208–212.

[12] Jose Luis Flores Campana, Luís Gustavo Lorgus Decker, Marcos Roberto e Souza, Helena de Almeida Maia, and Helio Pedrini. "Variable-hyperparameter visual transformer for efficient image inpainting." In: *Computers & Graphics* 113 (2023), pp. 57–68. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2023.05.006. URL: https://www.sciencedirect.com/science/article/pii/S0097849323000614.

[13] Andrew Campbell, Joe Benton, Valentin De Bortoli, Tom Rainforth, George Deligiannidis, and Arnaud Doucet. *A Continuous Time Framework for Discrete Denoising Models*. 2022. arXiv: 2205.14987 [stat.ML]. URL: https://arxiv.org/abs/2205.14987.

[14] Ya-Liang Chang, Zhe Yu Liu, Kuan-Ying Lee, and Winston Hsu. *Free-form Video Inpainting with 3D Gated Convolution and Temporal PatchGAN*. 2019. arXiv: 1904.10247 [cs.CV]. URL: https://arxiv.org/abs/1904.10247.

[15] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. *Neural Ordinary Differential Equations*. 2019. arXiv: 1806.07366 [cs.LG]. URL: https://arxiv.org/abs/1806.07366.

[16] Nicolas Cherel. "Internal methods for the generation and inpainting of images and videos." Theses. Institut Polytechnique de Paris, Mar. 2024. URL: https://theses.hal.science/tel-04573417.

[17] Nicolas Cherel, Andrés Almansa, Yann Gousseau, and Alasdair Newson. *Diffusion-based image inpainting with internal learning*. 2024. arXiv: 2406.04206 [cs.CV]. URL: https://arxiv.org/abs/2406.04206.

[18] Lu Chi, Borui Jiang, and Yadong Mu. "Fast Fourier Convolution." In: *Neural Information Processing Systems*. 2020. URL: https://api.semanticscholar.org/CorpusID:227276693.

[19] A. Criminisi, P. Perez, and K. Toyama. "Region filling and object removal by exemplar-based image inpainting." In: *IEEE Transactions on Image Processing* 13.9 (2004), pp. 1200–1212. DOI: 10.1109/TIP.2004.833105.

[20] Marco Cuturi. *Sinkhorn Distances: Lightspeed Computation of Optimal Transportation Distances*. 2013. arXiv: 1306.0895 [stat.ML]. URL: https://arxiv.org/abs/1306.0895.

[21] Giannis Daras, Hyungjin Chung, Chieh-Hsin Lai, Yuki Mitsufuji, Jong Chul Ye, Peyman Milanfar, Alexandros G. Dimakis, and Mauricio Delbracio. *A Survey on Diffusion Models for Inverse Problems*. 2024. arXiv: 2410.00083 [cs.LG]. URL: https://arxiv.org/abs/2410.00083.

[22] Ye Deng, Siqi Hui, Sanping Zhou, Deyu Meng, and Jinjun Wang. "T-former: An Efficient Transformer for Image Inpainting." In: *Proceedings of the 30th ACM International Conference on Multimedia*. MM '22. ACM, Oct. 2022, 6559–6568. DOI: 10.1145/3503161.3548446. URL: http://dx.doi.org/10.1145/3503161.3548446.

[23] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. URL: https://arxiv.org/abs/2010.11929.

[24] Iddo Drori, Daniel Cohen-Or, and Hezy Yeshurun. "Fragment-based image completion." In: *ACM Trans. Graph.* 22.3 (July 2003), 303–312. ISSN: 0730-0301. DOI: 10.1145/882262.882267. URL: https://doi.org/10.1145/882262.882267.

[25] Omar Elharrouss, Rafat Damseh, Abdelkader Nasreddine Belkacem, Elarbi Badidi, and Abderrahmane Lakas. "Transformer-based image and video inpainting: current challenges and future directions." In: *Artificial Intelligence Review* 58.4 (Feb. 2025). ISSN: 1573-7462. DOI: 10.1007/s10462-024-11075-9. URL: http://dx.doi.org/10.1007/s10462-024-11075-9.

[26] Patrick Esser et al. *Scaling Rectified Flow Transformers for High-Resolution Image Synthesis*. 2024. arXiv: 2403.03206 [cs.CV]. URL: https://arxiv.org/abs/2403.03206.

[27] Chen Gao, Ayush Saraf, Jia-Bin Huang, and Johannes Kopf. *Flow-edge Guided Video Completion*. 2020. arXiv: 2009.01835 [cs.CV]. URL: https://arxiv.org/abs/2009.01835.

[28] Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky T. Q. Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. *Discrete Flow Matching*. 2024. arXiv: 2407.15595 [cs.LG]. URL: https://arxiv.org/abs/2407.15595.

[29] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML]. URL: https://arxiv.org/abs/1406.2661.

[30] Miguel Granados, James Tompkin, {Kwang In} Kim, Oliver Grau, Jan Kautz, and Christian Theobalt. "How not to be seen: object removal from videos of crowded scenes." English. In: *Computer Graphics Forum* 31.2 Part 1 (May 2012), pp. 219–228. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2012.03000.x.

[31] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. *FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models*. 2018. arXiv: 1810.01367 [cs.LG]. URL: https://arxiv.org/abs/1810.01367.

[32] Christine Guillemot and Olivier Le Meur. "Image Inpainting : Overview and Recent Advances." In: *IEEE Signal Processing Magazine* 31.1 (2014), pp. 127–144. DOI: 10.1109/MSP.2013.2273004.

[33] Xintong Han, Zuxuan Wu, Weilin Huang, Matthew R. Scott, and Larry S. Davis. *Compatible and Diverse Fashion Image Inpainting*. 2019. arXiv: 1902.01096 [cs.CV]. URL: https://arxiv.org/abs/1902.01096.

[34] Jan Herling and Wolfgang Broll. "High-Quality Real-Time Video Inpaintingwith PixMix." In: *IEEE Transactions on Visualization and Computer Graphics* 20.6 (2014), pp. 866–879. DOI: 10.1109/TVCG.2014.2298016.

[35] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG]. URL: https://arxiv.org/abs/2006.11239.

[36] Jonathan Ho and Tim Salimans. *Classifier-Free Diffusion Guidance*. 2022. arXiv: 2207.12598 [cs.LG]. URL: https://arxiv.org/abs/2207.12598.

[37] Peter Holderrieth, Marton Havasi, Jason Yim, Neta Shaul, Itai Gat, Tommi Jaakkola, Brian Karrer, Ricky T. Q. Chen, and Yaron Lipman. *Generator Matching: Generative modeling with arbitrary Markov processes*. 2025. arXiv: 2410.20587 [cs.LG]. URL: https://arxiv.org/abs/2410.20587.

[38] Md Imran Hosen and Md Baharul Islam. *HiMFR: A Hybrid Masked Face Recognition Through Face Inpainting*. 2022. arXiv: 2209.08930 [cs.CV]. URL: https://arxiv.org/abs/2209.08930.

[39] Yuan-Ting Hu, Heng Wang, Nicolas Ballas, Kristen Grauman, and Alexander Schwing. "Proposal-Based Video Completion." In: Nov. 2020, pp. 38–54. ISBN: 978-3-030-58582-2. DOI: 10.1007/978-3-030-58583-9_3.

[40] Jia-Bin Huang, Sing Bing Kang, Narendra Ahuja, and Johannes Kopf. "Temporally coherent completion of dynamic video." In: *ACM Trans. Graph.* 35.6 (Nov. 2016). ISSN: 0730-0301. DOI: 10.1145/2980179.2982398. URL: https://doi.org/10.1145/2980179.2982398.

[41] Jireh Jam, Connah Kendrick, Kevin Walker, Vincent Drouard, Jison Gee-Sern Hsu, and Moi Hoon Yap. "A comprehensive review of past and present image inpainting methods." In: *Computer Vision and Image Understanding* 203 (2021), p. 103147. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2020.103147. URL: https://www.sciencedirect.com/science/article/pii/S1077314220301661.

[42] Ulugbek S. Kamilov, Charles A. Bouman, Gregery T. Buzzard, and Brendt Wohlberg. "Plug-and-Play Methods for Integrating Physical and Learned Models in Computational Imaging: Theory, algorithms, and applications." In: *IEEE Signal Processing Magazine* 40.1 (Jan. 2023), 85–97. ISSN: 1558-0792. DOI: 10.1109/msp.2022.3199595. URL: http://dx.doi.org/10.1109/MSP.2022.3199595.

[43] Dahun Kim, Sanghyun Woo, Joon-Young Lee, and In So Kweon. *Deep Video Inpainting*. 2019. arXiv: 1905.01639 [cs.CV]. URL: https://arxiv.org/abs/1905.01639.

[44] Soohyun Kim, Jongbeom Baek, Jihye Park, Gyeongnyeon Kim, and Seungryong Kim. *InstaFormer: Instance-Aware Image-to-Image Translation with Transformer*. 2022. arXiv: 2203.16248 [cs.CV]. URL: https://arxiv.org/abs/2203.16248.

[45] Diederik P. Kingma and Prafulla Dhariwal. *Glow: Generative Flow with Invertible 1x1 Convolutions*. 2018. arXiv: 1807.03039 [stat.ML]. URL: https://arxiv.org/abs/1807.03039.

[46] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML]. URL: https://arxiv.org/abs/1312.6114.

[47] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. "Normalizing Flows: An Introduction and Review of Current Methods." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.11 (Nov. 2021), 3964–3979. ISSN: 1939-3539. DOI: 10.1109/tpami.2020.2992934. URL: http://dx.doi.org/10.1109/TPAMI.2020.2992934.

[48] Nikita Kornilov, Petr Mokrov, Alexander Gasnikov, and Alexander Korotin. *Optimal Flow Matching: Learning Straight Trajectories in Just One Step*. 2024. arXiv: 2403.13117 [stat.ML]. URL: https://arxiv.org/abs/2403.13117.

[49] Thuc Trinh Le, Andrés Almansa, Yann Gousseau, and Simon Masnou. "Motion-consistent video inpainting." In: *2017 IEEE International Conference on Image Processing (ICIP)*. 2017, pp. 2094–2098. DOI: 10.1109/ICIP.2017.8296651.

[50] Sungho Lee, Seoung Wug Oh, DaeYeun Won, and Seon Joo Kim. *Copy-and-Paste Networks for Deep Video Inpainting*. 2019. arXiv: 1908.11587 [cs.CV]. URL: https://arxiv.org/abs/1908.11587.

[51] Wenbo Li, Zhe Lin, Kun Zhou, Lu Qi, Yi Wang, and Jiaya Jia. *MAT: Mask-Aware Transformer for Large Hole Image Inpainting*. 2022. arXiv: 2203.15270 [cs.CV]. URL: https://arxiv.org/abs/2203.15270.

[52] Zhen Li, Cheng-Ze Lu, Jianhua Qin, Chun-Le Guo, and Ming-Ming Cheng. *Towards An End-to-End Framework for Flow-Guided Video Inpainting*. 2022. arXiv: 2204.02663 [eess.IV]. URL: https://arxiv.org/abs/2204.02663.

[53] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. *Flow Matching for Generative Modeling*. 2023. arXiv: 2210.02747 [cs.LG]. URL: https://arxiv.org/abs/2210.02747.

[54] Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky T. Q. Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. *Flow Matching Guide and Code*. 2024. arXiv: 2412.06264 [cs.LG]. URL: https://arxiv.org/abs/2412.06264.

[55] Guan-Horng Liu, Arash Vahdat, De-An Huang, Evangelos A. Theodorou, Weili Nie, and Anima Anandkumar. *I^2SB: Image-to-Image Schrödinger Bridge*. 2023. arXiv: 2302.05872 [cs.CV]. URL: https://arxiv.org/abs/2302.05872.

[56] Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. *Image Inpainting for Irregular Holes Using Partial Convolutions*. 2018. arXiv: 1804.07723 [cs.CV]. URL: https://arxiv.org/abs/1804.07723.

[57] Hongyu Liu, Ziyu Wan, Wei Huang, Yibing Song, Xintong Han, and Jing Liao. "PD-GAN: Probabilistic Diverse GAN for Image Inpainting." In: June 2021, pp. 9367–9376. DOI: 10.1109/CVPR46437.2021.00925.

[58] Qiang Liu. *Rectified Flow: A Marginal Preserving Approach to Optimal Transport*. 2022. arXiv: 2209.14577 [stat.ML]. URL: https://arxiv.org/abs/2209.14577.

[59] Xingchao Liu, Chengyue Gong, and Qiang Liu. *Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow*. 2022. arXiv: 2209.03003 [cs.LG]. URL: https://arxiv.org/abs/2209.03003.

[60] Xingchao Liu, Xiwen Zhang, Jianzhu Ma, Jian Peng, and Qiang Liu. *InstaFlow: One Step is Enough for High-Quality Diffusion-Based Text-to-Image Generation*. 2024. arXiv: 2309.06380 [cs.LG]. URL: https://arxiv.org/abs/2309.06380.

[61] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV]. URL: https://arxiv.org/abs/2103.14030.

[62] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG]. URL: https://arxiv.org/abs/1711.05101.

[63] Andreas Lugmayr, Martin Danelljan, Luc Van Gool, and Radu Timofte. *SRFlow: Learning the Super-Resolution Space with Normalizing Flow*. 2020. arXiv: 2006.14200 [cs.CV]. URL: https://arxiv.org/abs/2006.14200.

[64] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. *RePaint: Inpainting using Denoising Diffusion Probabilistic Models*. 2022. arXiv: 2201.09865 [cs.CV]. URL: https://arxiv.org/abs/2201.09865.

[65] Y. Matsushita, E. Ofek, Weina Ge, Xiaoou Tang, and Heung-Yeung Shum. "Full-frame video stabilization with motion inpainting." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.7 (2006), pp. 1150–1163. DOI: 10.1109/TPAMI.2006.141.

[66] Robert J. McCann. "A Convexity Principle for Interacting Gases." In: *Advances in Mathematics* 128.1 (1997), pp. 153–179. ISSN: 0001-8708. DOI: https://doi.org/10.1006/aima.1997.1634. URL: https://www.sciencedirect.com/science/article/pii/S0001870897916340.

[67] Rito Murase, Yan Zhang, and Takayuki Okatani. "Video-Rate Video Inpainting." In: Jan. 2019, pp. 1553–1561. DOI: 10.1109/WACV.2019.00170.

[68] MohammadReza Naderi, MohammadHossein Givkashi, Nader Karimi, Shahram Shirani, and Shadrokh Samavi. *SFI-Swin: Symmetric Face Inpainting with Swin Transformer by Distinctly Learning Face Components Distributions*. 2023. arXiv: 2301.03130 [cs.CV]. URL: https://arxiv.org/abs/2301.03130.

[69] Alasdair Newson, Andrés Almansa, Matthieu Fradet, Yann Gousseau, and Patrick Perez. "Vers un inpainting vidéo automatique, rapide et générique." In: *(Gretsi 2013) 23ème Colloque Gretsi*. Brest, France, 2013. URL: https://telecom-paris.hal.science/hal-02287085.

[70] Alasdair Newson, Andrés Almansa, Matthieu Fradet, Yann Gousseau, and Patrick Pérez. "Video Inpainting of Complex Scenes." In: *SIAM Journal on Imaging Sciences* 7.4 (Jan. 2014), 1993–2019. ISSN: 1936-4954. DOI: 10.1137/140954933. URL: http://dx.doi.org/10.1137/140954933.

[71] Alasdair Newson, Andrés Almansa, Yann Gousseau, and Patrick Pérez. "Non-Local Patch-Based Image Inpainting." In: *Image Processing On Line* 7 (2017). https://doi.org/10.5201/ipol.2017.189, pp. 373–385.

[72] Alex Nichol and Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. 2021. arXiv: 2102.09672 [cs.LG]. URL: https://arxiv.org/abs/2102.09672.

[73] Seoung Wug Oh, Sungho Lee, Joon-Young Lee, and Seon Joo Kim. *Onion-Peel Networks for Deep Video Completion*. 2019. arXiv: 1908.08718 [cs.CV]. URL: https://arxiv.org/abs/1908.08718.

[74] K.A. Patwardhan, G. Sapiro, and M. Bertalmio. "Video inpainting of occluding and occluded objects." In: *IEEE International Conference on Image Processing 2005*. Vol. 2. 2005, pp. II–69. DOI: 10.1109/ICIP.2005.1529993.

[75] Kedar A. Patwardhan, Guillermo Sapiro, and Marcelo Bertalmio. "Video Inpainting Under Constrained Camera Motion." In: *IEEE Transactions on Image Processing* 16.2 (2007), pp. 545–553. DOI: 10.1109/TIP.2006.888343.

[76] Gabriel Peyré and Marco Cuturi. *Computational Optimal Transport*. 2020. arXiv: 1803.00567 [stat.ML]. URL: https://arxiv.org/abs/1803.00567.

[77] Aram-Alexandre Pooladian, Heli Ben-Hamu, Carles Domingo-Enrich, Brandon Amos, Yaron Lipman, and Ricky T. Q. Chen. *Multisample Flow Matching: Straightening Flows with Minibatch Couplings*. 2023. arXiv: 2304.14772 [cs.LG]. URL: https://arxiv.org/abs/2304.14772.

[78] Weize Quan, Jiaxi Chen, Yanli Liu, Dong-Ming Yan, and Peter Wonka. *Deep Learning-based Image and Video Inpainting: A Survey*. 2024. arXiv: 2401.03395 [cs.CV]. URL: https://arxiv.org/abs/2401.03395.

[79] Danilo Jimenez Rezende and Shakir Mohamed. *Variational Inference with Normalizing Flows*. 2016. arXiv: 1505.05770 [stat.ML]. URL: https://arxiv.org/abs/1505.05770.

[80] Danilo Jimenez Rezende and Shakir Mohamed. *Variational Inference with Normalizing Flows*. 2016. arXiv: 1505.05770 [stat.ML]. URL: https://arxiv.org/abs/1505.05770.

[81] Herbert E. Robbins. "An Empirical Bayes Approach to Statistics." In: 1956. URL: https://api.semanticscholar.org/CorpusID:26161481.

[82] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2022. arXiv: 2112.10752 [cs.CV]. URL: https://arxiv.org/abs/2112.10752.

[83] Noam Rozen, Aditya Grover, Maximilian Nickel, and Yaron Lipman. *Moser Flow: Divergence-based Generative Modeling on Manifolds*. 2021. arXiv: 2108.08052 [stat.ML]. URL: https://arxiv.org/abs/2108.08052.

[84] Min-cheol Sagong, Yong-goo Shin, Seung-wook Kim, Seung Park, and Sung-jea Ko. "PEPSI : Fast Image Inpainting With Parallel Decoding Network." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 11352–11360. DOI: 10.1109/CVPR.2019.01162.

[85] Chitwan Saharia, William Chan, Huiwen Chang, Chris A. Lee, Jonathan Ho, Tim Salimans, David J. Fleet, and Mohammad Norouzi. *Palette: Image-to-Image Diffusion Models*. 2022. arXiv: 2111.05826 [cs.CV]. URL: https://arxiv.org/abs/2111.05826.

[86] Tim Salimans and Jonathan Ho. *Progressive Distillation for Fast Sampling of Diffusion Models*. 2022. arXiv: 2202.00512 [cs.LG]. URL: https://arxiv.org/abs/2202.00512.

[87] Joana Cristo Santos, Hugo Tomás Pereira Alexandre, Miriam Seoane Santos, and Pedro Henriques Abreu. "The Role of Deep Learning in Medical Image Inpainting: A Systematic Review." In: *ACM Trans. Comput. Healthcare* 6.3 (May 2025). DOI: 10.1145/3712710. URL: https://doi.org/10.1145/3712710.

[88] T. Shiratori, Y. Matsushita, Xiaoou Tang, and Sing Bing Kang. "Video Completion by Motion Field Transfer." In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 1. 2006, pp. 411–418. DOI: 10.1109/CVPR.2006.330.

[89] Uriel Singer et al. *Make-A-Video: Text-to-Video Generation without Text-Video Data*. 2022. arXiv: 2209.14792 [cs.CV]. URL: https://arxiv.org/abs/2209.14792.

[90] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 2015. arXiv: 1503.03585 [cs.LG]. URL: https://arxiv.org/abs/1503.03585.

[91] Jiaming Song, Chenlin Meng, and Stefano Ermon. *Denoising Diffusion Implicit Models*. 2022. arXiv: 2010.02502 [cs.LG]. URL: https://arxiv.org/abs/2010.02502.

[92] Yang Song and Stefano Ermon. *Generative Modeling by Estimating Gradients of the Data Distribution*. 2020. arXiv: 1907.05600 [cs.LG]. URL: https://arxiv.org/abs/1907.05600.

[93] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. *Score-Based Generative Modeling through Stochastic Differential Equations*. 2021. arXiv: 2011.13456 [cs.LG]. URL: https://arxiv.org/abs/2011.13456.

[94] Michael Strobel, Julia Diebold, and Daniel Cremers. "Flow and color inpainting for video completion." English. In: *Pattern Recognition - 36th German Conference, GCPR 2014, Proceedings*. Ed. by Joachim Hornegger, Xiaoyi Jiang, Joachim Hornegger, Joachim Hornegger, and Reinhard Koch. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Publisher Copyright: © Springer International Publishing Switzerland 2014.; 36th German Conference on Pattern Recognition, GCPR 2014 ; Conference date: 02-09-2014 Through 05-09-2014. Springer Verlag, 2014, pp. 293–304. DOI: 10.1007/978-3-319-11752-2_23.

[95] Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. *Resolution-robust Large Mask Inpainting with Fourier Convolutions*. 2021. arXiv: 2109.07161 [cs.CV]. URL: https://arxiv.org/abs/2109.07161.

[96] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains*. 2020. arXiv: 2006.10739 [cs.CV]. URL: https://arxiv.org/abs/2006.10739.

[97] Matias Tassano, Julie Delon, and Thomas Veit. *FastDVDnet: Towards Real-Time Deep Video Denoising Without Flow Estimation*. 2020. arXiv: 1907.01361 [cs.CV]. URL: https://arxiv.org/abs/1907.01361.

[98] Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. *Improving and generalizing flow-based generative models with minibatch optimal transport*. 2024. arXiv: 2302.00482 [cs.LG]. URL: https://arxiv.org/abs/2302.00482.

[99] Alexander Tong, Jessie Huang, Guy Wolf, David van Dijk, and Smita Krishnaswamy. *TrajectoryNet: A Dynamic Optimal Transport Network for Modeling Cellular Dynamics*. 2020. arXiv: 2002.04461 [stat.ML]. URL: https://arxiv.org/abs/2002.04461.

[100] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[101] Cédric Villani. "Optimal transport – Old and new." In: vol. 338. Jan. 2008, pp. xxii+973. DOI: 10.1007/978-3-540-71050-9.

[102] Apoorv Vyas et al. *Audiobox: Unified Audio Generation with Natural Language Prompts*. 2023. arXiv: 2312.15821 [cs.SD]. URL: https://arxiv.org/abs/2312.15821.

[103] Ziyu Wan, Jingbo Zhang, Dongdong Chen, and Jing Liao. *High-Fidelity Pluralistic Image Completion with Transformers*. 2021. arXiv: 2103.14031 [cs.CV]. URL: https://arxiv.org/abs/2103.14031.

[104] Cairong Wang, Yiming Zhu, and Chun Yuan. "Diverse Image Inpainting with Normalizing Flow." In: *Computer Vision – ECCV 2022*. Ed. by Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner. Cham: Springer Nature Switzerland, 2022, pp. 53–69. ISBN: 978-3-031-20050-2.

[105] Chuan Wang, Haibin Huang, Xiaoguang Han, and Jue Wang. *Video Inpainting by Jointly Learning Temporal Structure and Spatial Details*. 2018. arXiv: 1806.08482 [cs.CV]. URL: https://arxiv.org/abs/1806.08482.

[106] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. *Non-local Neural Networks*. 2018. arXiv: 1711.07971 [cs.CV]. URL: https://arxiv.org/abs/1711.07971.

[107] Yonatan Wexler, Eli Shechtman, and Michal Irani. "Space-Time Completion of Video." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.3 (2007), pp. 463–476. DOI: 10.1109/TPAMI.2007.60.

[108] Weihao Xia, Yulun Zhang, Yujiu Yang, Jing-Hao Xue, Bolei Zhou, and Ming-Hsuan Yang. "GAN Inversion: A Survey." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2022).

[109] Hanyu Xiang, Qin Zou, Muhammad Nawaz, Xianfeng Huang, Fan Zhang, and Hongkai Yu. "Deep Learning for Image Inpainting: A Survey." In: *Pattern Recognition* 134 (Sept. 2022), p. 109046. DOI: 10.1016/j.patcog.2022.109046.

[110] Jianwen Xie, Song-Chun Zhu, and Ying Nian Wu. *Synthesizing Dynamic Patterns by Spatial-Temporal Generative ConvNet*. 2017. arXiv: 1606.00972 [stat.ML]. URL: https://arxiv.org/abs/1606.00972.

[111] Rui Xu, Xiaoxiao Li, Bolei Zhou, and Chen Change Loy. *Deep Flow-Guided Video Inpainting*. 2019. arXiv: 1905.02884 [cs.CV]. URL: https://arxiv.org/abs/1905.02884.

[112] Zili Yi, Qiang Tang, Shekoofeh Azizi, Daesik Jang, and Zhan Xu. *Contextual Residual Aggregation for Ultra High-Resolution Image Inpainting*. 2020. arXiv: 2005.09704 [cs.CV]. URL: https://arxiv.org/abs/2005.09704.

[113] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. *Generative Image Inpainting with Contextual Attention*. 2018. arXiv: 1801.07892 [cs.CV]. URL: https://arxiv.org/abs/1801.07892.

[114] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas Huang. *Free-Form Image Inpainting with Gated Convolution*. 2019. arXiv: 1806.03589 [cs.CV]. URL: https://arxiv.org/abs/1806.03589.

[115] Yanhong Zeng, Jianlong Fu, and Hongyang Chao. *Learning Joint Spatial-Temporal Transformations for Video Inpainting*. 2020. arXiv: 2007.10247 [cs.CV]. URL: https://arxiv.org/abs/2007.10247.

[116] Kaidong Zhang, Jingjing Fu, and Dong Liu. *Flow-Guided Transformer for Video Inpainting*. 2022. arXiv: 2208.06768 [cs.CV]. URL: https://arxiv.org/abs/2208.06768.

[117] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. *Adding Conditional Control to Text-to-Image Diffusion Models*. 2023. arXiv: 2302.05543 [cs.CV]. URL: https://arxiv.org/abs/2302.05543.

[118] Shengyu Zhao, Jonathan Cui, Yilun Sheng, Yue Dong, Xiao Liang, Eric I Chang, and Yan Xu. *Large Scale Image Completion via Co-Modulated Generative Adversarial Networks*. 2021. arXiv: 2103.10428 [cs.CV]. URL: https://arxiv.org/abs/2103.10428.

[119] Chuanxia Zheng, Tat-Jen Cham, and Jianfei Cai. *Pluralistic Image Completion*. 2019. arXiv: 1903.04227 [cs.CV]. URL: https://arxiv.org/abs/1903.04227.

[120] Qinqing Zheng, Matt Le, Neta Shaul, Yaron Lipman, Aditya Grover, and Ricky T. Q. Chen. *Guided Flows for Generative Modeling and Decision Making*. 2023. arXiv: 2311.13443 [cs.LG]. URL: https://arxiv.org/abs/2311.13443.

[121] Zangwei Zheng, Xiangyu Peng, Tianji Yang, Chenhui Shen, Shenggui Li, Hongxin Liu, Yukun Zhou, Tianyi Li, and Yang You. *Open-Sora: Democratizing Efficient Video Production for All*. 2024. arXiv: 2412.20404 [cs.CV]. URL: https://arxiv.org/abs/2412.20404.

[122] Shangchen Zhou, Chongyi Li, Kelvin C. K. Chan, and Chen Change Loy. *ProPainter: Improving Propagation and Transformer for Video Inpainting*. 2023. arXiv: 2309.03897 [cs.CV]. URL: https://arxiv.org/abs/2309.03897.