Rapport de Projet de Fin d'Études

Laurent Fainsin
laurent@fainsin.bzh

Département Sciences du Numérique
Troisième année
2022 — 2023

Remerciements

Je tiens à remercier Xavier Roynard, Michele Alessandro Bucci et Brian Staber, mes tuteurs de stage, ainsi que les équipes de Safran pour leur accueil et leur accompagnement tout au long de ce stage.

Je tiens à remercier ma famille pour le soutien qu'elle m'a apporté tout au long de mon stage et plus généralement dans ma vie.

J'aimerais également remercier l'ensemble de mes professeurs de l'École nationale supérieure d'électrotechnique, d'électronique, d'informatique, d'hydraulique et des télécommunications (ENSEEIHT), pour m'avoir permis d'acquérir les connaissances nécessaires à la réalisation de ce projet.

Table des matières

Table des figures

Abbréviations et acronymes

1-NNA 1-Nearest Neighbor Accuracy. iii, 8, 9

ARM Auto-Regressive Model. iv, v, 18

CAO Conception Assistée par Ordinateur. 5, 35

CD Chamfer Distance. iii, 7, 9

CFD Computational Fluid Dynamic. 3, 21, 22, 32, 33

CFG Classifier-Free Guidance. iv, v, 17, 18, 31

CG Classifier Guidance. v, 16

COV Coverage. iii, 8, 9

DDPM Denoising Diffusion Probabilistic Model. 13, 14, 17, 18, 27, 28, 35

DST Digital Sciences and Technologies. 20

EDP Équation aux Dérivées Partielles. 5

ELBO Evidence Lower Bound. 11, 13, 14

EMD Earth Mover Distance. iii, 7, 9

ENSEEIHT École nationale supérieure d'électrotechnique, d'électronique, d'informatique, d'hydraulique et des télécommunications. ii

FLEX physics inFormed machine Learning and numerical EXploration. 20

GAN Generative Adversarial Network. iii, v, 4, 9, 10, 11

GNN Graph Neural Network. iii, 4, 5, 6

Go Gigaoctet. 22

GP Gaussian Process. iv, vi, 32, 33, 34, 35

HD Haussdorf Distance. iii, 7

JSD Jensen-Shannon Divergence. iii, 8, 9

KLD Kullback-Leibler Divergence. iii, 7, 8, 11, 13

KP-CNN Kernel Point Convolutional Neural Network. v, 26, 27

KP-FCNN Kernel Point Fully Convolutional Neural Network. iv, v, 26, 27, 29, 31

KPConv Kernel Point Convolution. iv, v, 26, 27

LDM Latent Diffusion Model. v, vi, 15, 30, 31, 32
LION Latent Point Diffusion Model. iv, vi, 29, 30, 31

MADS Mathematics Applied to Design and Simulation. 20
MLP Multi-Layer Perceptron. 24, 25, 27, 28, 31
MMD Minimum Matching Distance. iii, 8, 9

NeRF Neural Radiance Field. iv, v, 18, 19
NF Normalizing Flow. iii, v, 4, 11, 12

PCA Principal Component Analysis. vi, 30, 31, 32, 35
POD Proper Orthogonal Decomposition. 31
PVConv Point Voxel Convolution. iv, vi, 27, 28
PVD Point Voxel Diffusion. iv, vi, 28, 29, 31

RL Reinforcement Learning. 35

SPVConv Sparse Point Voxel Convolution. 28

T2I Text to Image. 15

VAE Variational Auto-Encoder. iii, v, 4, 10, 11, 12, 13, 15, 23, 24, 29
VDM Variational Diffusion Model. iv, v, 4, 12, 13, 14, 15, 27

Chapitre 1

Introduction

1.1 Présentation de l'entreprise

Safran est un grand groupe industriel et technologique français, présent au niveau international dans les domaines de l'aéronautique, de l'espace et de la défense. Safran est le fruit de la fusion de Snecma et Sagem en 2005 et de l'acquisition de Zodiac Aerospace en 2018.

Les sociétés du groupe (Safran Aero Boosters, Safran Aerosystems, Safran Aircraft Engines, Safran Cabin, Safran Ceramics, Safran Electrical & Power, Safran Electronics & Defense, Safran Filtration System, Safran Helicopter Engines, Safran Landing Systems, Safran Nacelles, Safran Passenger Solutions, Safran Seats et Safran Transmission Systems) se répartissent entre cinq secteurs d'activité, à savoir la propulsion aéronautique et spatiale, les équipements aéronautiques, les intérieurs d'avions, les aérosystèmes et les applications militaires.

Dans le secteur des moteurs destinés aux avions civils, militaires et hélicoptères, Safran fait face à quatre concurrents majeurs: General Electric des États-Unis, Rolls-Royce du Royaume-Uni, Pratt & Whitney des États-Unis et du Canada, ainsi que Honeywell des États-Unis. Dans le domaine des équipements, la compétition est plus vaste et variée:

- Pour les systèmes d'atterrissage: Collins Aerospace (États-Unis), General Electric (États-Unis) et Crane Aerospace & Electronics (États-Unis).
- En ce qui concerne les roues et les freins: Collins Aerospace, Honeywell (États-Unis) et Meggitt (Royaume-Uni).
- Pour les nacelles: Collins Aerospace et Spirit AeroSystems (États-Unis).

Dans les domaines de la défense et de l'avionique, le principal concurrent est le groupe français Thales.

Par ailleurs, Safran s'est également investi dans le développement de drones tactiques, tels que le Sagem Sperwer et le Patroller. Dans ce secteur, les principaux concurrents de Safran incluent AAI Corp (États-Unis), IAI (Israël) et Thales.

Safran est une société anonyme, côté au CAC 40 depuis 2011, l'État Français en possède 11% et ses 83 000 salariés 7%. Une grande part de son développement économique repose sur la vente de moteurs pour l'aviation civile. Son chiffre d'affaire s'élève à 10 945 millions d'euros au S1 2023.

Il est possible d'identifier des cycles itératifs comprenant les phases de recherche, de production et de vente, avec une focalisation sur le développement des moteurs. Une fois que la phase de recherche aboutit à la conception du moteur, celui-ci entre en production. À ce stade, l'entreprise se concentre sur l'amélioration des processus de production et leur mise à l'échelle. Les ventes génèrent des revenus pour l'entreprise, qui sont ensuite réinvestis dans le développement du moteur suivant.

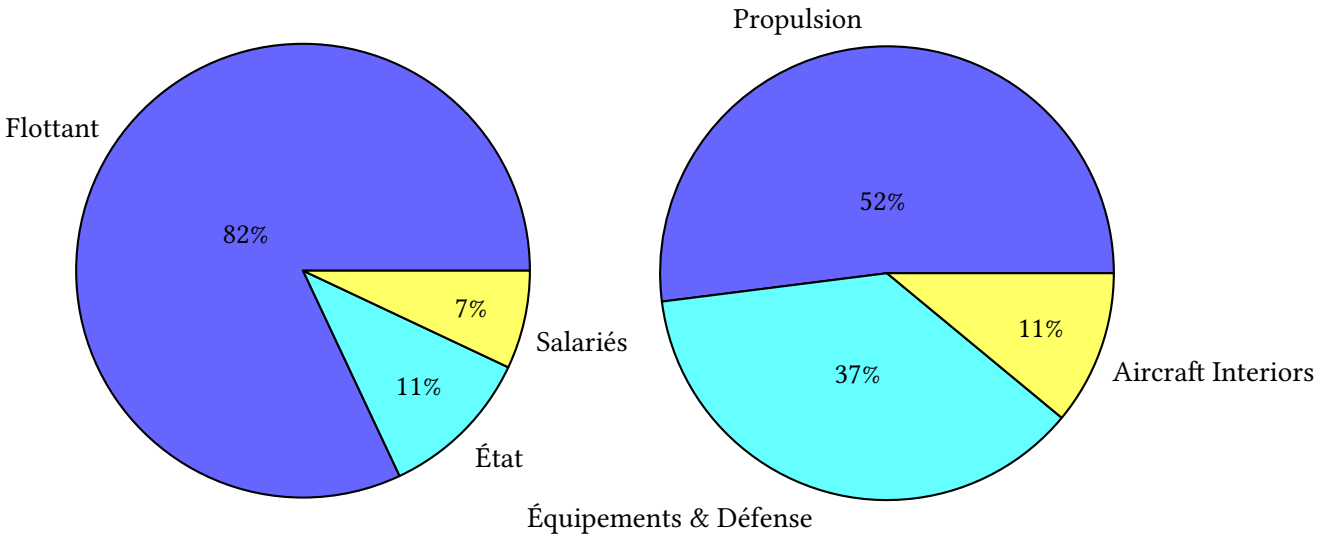


FIGURE 1.1 – Gauche: Structure du capital au S1 2023. Droite: Répartition du chiffre d'affaires au S1 2023.

FIGURE 1.2 – Gauche: CFM56-7B. Droite: M88.
Copyright Eric Drouin / Philippe Stroppa / Safran.

Le CFM56, dont le développement s'est achevé en 1978, détient toujours le record du moteur civil le plus vendu au monde. Les ventes de ce moteur continuent à soutenir l'activité de Safran jusqu'à ce jour. En parallèle, le moteur LEAP, annoncé en 2008, a été mis sur le marché en 2014. Il équipe la nouvelle génération d'avions tels que les Boeing 737 Max et les Airbus A320 neo. Pour maintenir l'expertise technique malgré les fluctuations entre les cycles de développement et de production, Safran a créé Safran Tech en 2015. Cette entité prépare d'ores et déjà les technologies qui seront utilisées entre 2025 et 2030.

Safran Tech représente le centre de Recherche et Technologie de l'Innovation au sein du groupe. Son fonctionnement diffère du modèle commercial classique puisqu'il n'est pas axé sur un marché spécifique. Il abrite une équipe d'environ 500 chercheurs et ingénieurs qui se répartissent dans divers pôles, tels que "Matériaux & Procédés", "Énergie & Propulsion", "Systèmes Électriques et Électroniques", ainsi que "Sciences et Technologies du Numérique". Sa mission principale est de stimuler l'innovation sur des problématiques ayant un impact sur l'ensemble des filiales du groupe.

Les équipes de recherche au sein de Safran Tech ne sont pas directement soumises aux contraintes imposées

par d'importants projets industriels. Par conséquent, elles ont la liberté de mener des projets de recherche tout en collaborant avec des laboratoires externes tels que l'ONERA, le Centre des Matériaux des Mines, ainsi que les équipes d'autres entreprises. Cette structure permet à Safran Tech de rester agile dans ses activités de recherche et de développement, en explorant des solutions novatrices sans les contraintes de projets industriels spécifiques.

1.2 Contexte du stage

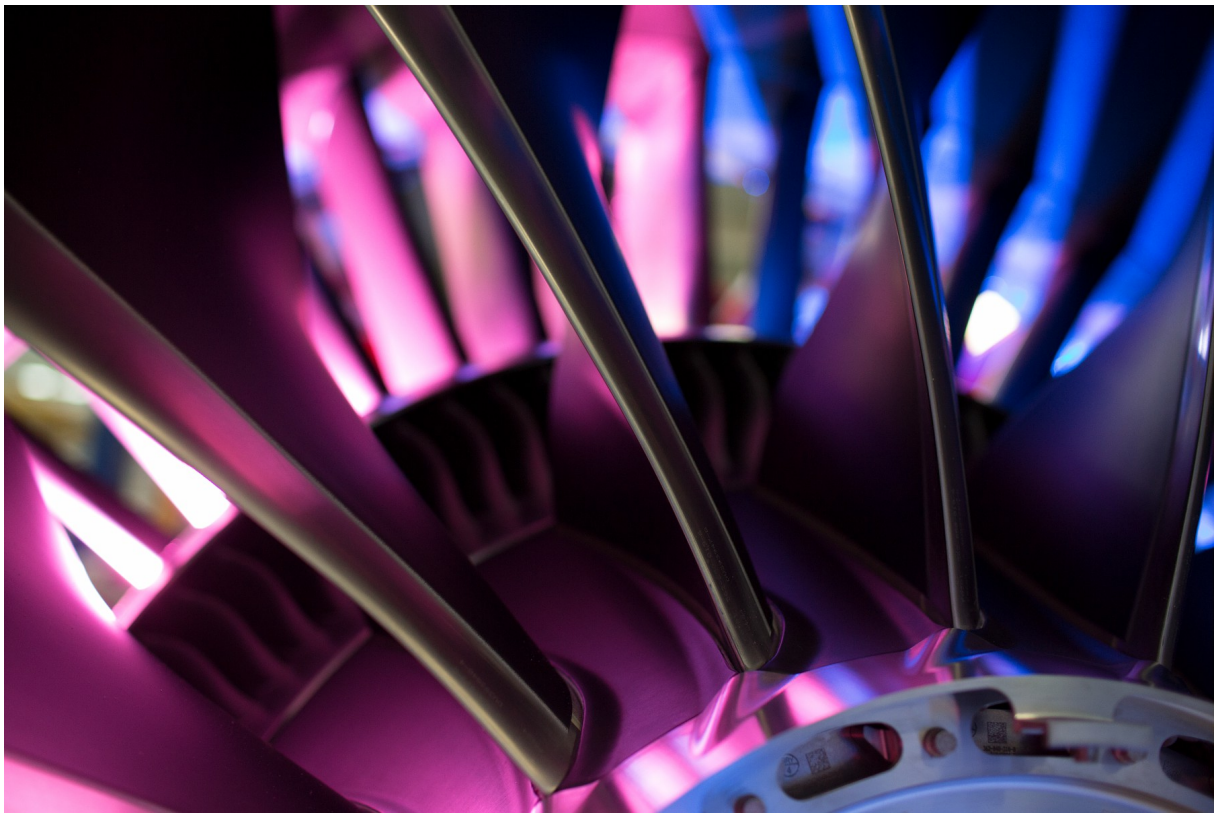


FIGURE 1.3 – Aubes du moteur Leap-1A.
Copyright Cyril Abad / CAPA Pictures / Safran.

Dans le domaine industriel, les codes de simulation numérique sont désormais un outil indispensable pour la conception de systèmes complexes, en particulier pour les modules de réacteurs d'avions ou d'hélicoptères.

De telles simulations sont par exemple utilisées pour évaluer les performances aérodynamiques d'un composant tel qu'une aube de turbine. En partant d'une géométrie nominale, dans la phase d'optimisation, la pièce est progressivement modifiée afin d'optimiser certaines quantités d'intérêt.

Malheureusement, ce processus de conception itératif présente deux limites:

- Le coût de calcul d'une simulation numérique de type Computational Fluid Dynamic (CFD) est lourd, plusieurs heures sont nécessaires pour un unique calcul.
- Le nombre de degrés de liberté pour la géométrie d'un profil complexe discrétisée avec un maillage non structuré est important, ce qui rend impossible l'exploration complète de l'espace de recherche de la solution optimale.

Les approches d'optimisation assistées par surfaces de réponse permettent de répondre partiellement à ces difficultés. Cependant cette stratégie admet deux limitations intrinsèques:

— Elles nécessitent un long de travail de paramétrisation.
— Elles souffrent grandement du fléau de la dimension (i.e. la taille des problèmes considérés est généralement limitée).

Toutefois, il est important de noter que les profils capables de générer de la portance, et par conséquent, de la puissance pour le moteur, présentent des similitudes marquées entre eux. Ces similitudes se manifestent généralement par de subtiles variations, principalement au niveau des bords d'attaque ou des bords de fuite. Cette observation conduit à l'hypothèse qu'une représentation latente parcimonieuse des aubes, qu'elles soient destinées à des turbines ou à des compresseurs, existe potentiellement. Cette représentation latente pourrait être exploitée pour explorer de manière plus efficiente l'espace de recherche en vue d'optimiser les quantités d'intérêt, ou pour vérifier la conformité aux contraintes de conception.

Récemment, les modèles génératifs profonds comme les Variational Auto-Encoders (VAEs) ou les Generative Adversarial Networks (GANs) ont été appliqués avec succès à des données structurées (e.g. des images). Ceux-ci permettent de construire un espace latent représentatif d'un jeu de données spécifique et de générer de nouveaux échantillons qui partagent des caractéristiques importantes du jeu de données d'entraînement.

Cependant, dans le cas de la simulation numérique, les données prennent souvent la forme de graphes en raison de l'utilisation de maillages pour représenter les surfaces des pièces à concevoir. Dans le contexte d'une application industrielle, il est donc crucial d'adapter les modèles susmentionnés afin de pouvoir utiliser des données non structurées en entrée. Les Graph Neural Networks (GNNs) permettent de traiter des données non structurées telles que des maillages ou des nuages de points.

Différentes solutions ont été proposées dans la littérature pour réaliser des convolutions et agrégations sur graphes ou nuages de points. Cependant, peu d'entre elles conviennent à l'application des réseaux sur graphes sur des données générées par des simulations numériques, car l'ordre de grandeur du nombre de nœuds est généralement trop important.

Le but de ce stage est d'évaluer le potentiel de ces nouvelles méthodes sur des jeux de données réalisés en internes et représentatifs pour Safran. Et éventuellement de proposer des améliorations spécifiques aux maillages utilisés en simulations numériques.

L'étude vise tout d'abord à étudier la bibliographie disponible d'un côté sur les modèles génératifs et d'un autre sur les réseaux convolutionnels sur graphes. L'objectif est, dans une première phase, de faire un benchmark des différentes solutions de modèles génératifs sur graphe de type VAE et GAN afin de créer une représentation latente des géométries d'aubes 3D. Pour cela un dataset avec quelques milliers d'échantillons d'aubes 3D et leurs performances aérodynamique est disponible à Safran. Le modèle résultant sera ensuite testé pour générer de nouvelles géométries et pour prédire les quantités d'intérêt par le biais de métamodèles classiques. Enfin, si l'avancement sur les premières tâches le permet, d'autres modèles génératifs peuvent être considérés comme les Normalizing Flows (NFs) ou les Variational Diffusion Models (VDMs).

Chapitre 2

État de l'art

Ce chapitre présente les différents concepts et méthodes nécessaires à la compréhension du travail réalisé durant ce stage.

Dans le cadre de cette étude, nous nous intéressons à la génération de géométries d'aubes de turbines par l'intermédiaire de maillages. Cette approche est motivée par plusieurs raisons, dont les considérations spécifiques suivantes:

- Les données que nous traitons sont destinées à alimenter des simulations numériques, où l'espace est discrétisé pour résoudre numériquement des Équation aux Dérivées Partielles (EDP) liées à la mécanique des solides ou des fluides.
- Les protocoles de Conception Assistée par Ordinateur (CAO), bien qu'efficaces pour la représentation géométrique, présentent des défis liés à la propriété des formats et aux différentes manières de représenter une même géométrie. Ceci peut être vu dans des logiciels tels que Catia, où diverses représentations de CAO peuvent coexister pour une géométrie donnée, rendant complexe la création d'un processus inverse.

Les maillages sont un domaine relativement peu exploré dans la littérature de l'apprentissage automatique, et cette exploration est encore plus limitée pour les représentations par CAO, en comparaison avec les modalités plus classiques telles que les images, le texte ou encore l'audio. La complexité découle en partie du caractère non structuré de ces données. En conséquence, il est nécessaire d'utiliser des méthodes spécifiques pour traiter ces données.

Il reste pertinent de noter que les méthodes présentées dans ce chapitre sont récentes et que la littérature évolue très rapidement. De plus, les méthodes existantes sont très nombreuses et il est impossible de toutes les présenter. Nous avons donc choisi de présenter les méthodes les plus pertinentes pour permettre une bonne compréhension globale du travail réalisé durant ce stage.

2.1 Graph Neural Network (GNN)

Les graphes sont des structures de données qui permettent de représenter des relations entre des entités. Un graphe est défini par un ensemble de nœuds et un ensemble d'arêtes. Les arêtes représentent des relations entre les nœuds. Ces relations peuvent être de différents types, comme des relations de parenté, de proximité ou encore de similarité. Les graphes peuvent être dirigés ou non. Dans le cas d'un graphe dirigé, les arêtes sont orientées et représentent une relation unidirectionnelle. Dans le cas d'un graphe non dirigé, les arêtes ne sont pas orientées et représentent une relation bidirectionnelle. Les graphes peuvent être pondérés ou non. Dans le cas d'un graphe pondéré, les arêtes sont associées à une valeur qui représente l'intensité de la relation entre

les nœuds.

Les graphes offrent une représentation intuitive de diverses structures, visibles dans la figure 2.1, telles que les réseaux de communication, les réseaux sociaux, les molécules ou encore les maillages. Par conséquent, les graphes sont un type de données largement présents dans la nature et sont très répandu dans le domaine de l'ingénierie. De manière générale, les graphes peuvent être considérés comme une généralisation des données structurées, telles que les images ou les séries temporelles. En effet, toute données structurées/régulières peut facilement être traduite en un graphe régulier.

Les maillages constituent une catégorie spécifique de graphes largement employée pour la représentation de surfaces. En plus des éléments caractéristiques des graphes, les maillages intègrent généralement des attributs associés à chaque triangle qui le compose. Ces attributs englobent des propriétés telles que les normales, les textures et les caractéristiques physiques, entre autres.

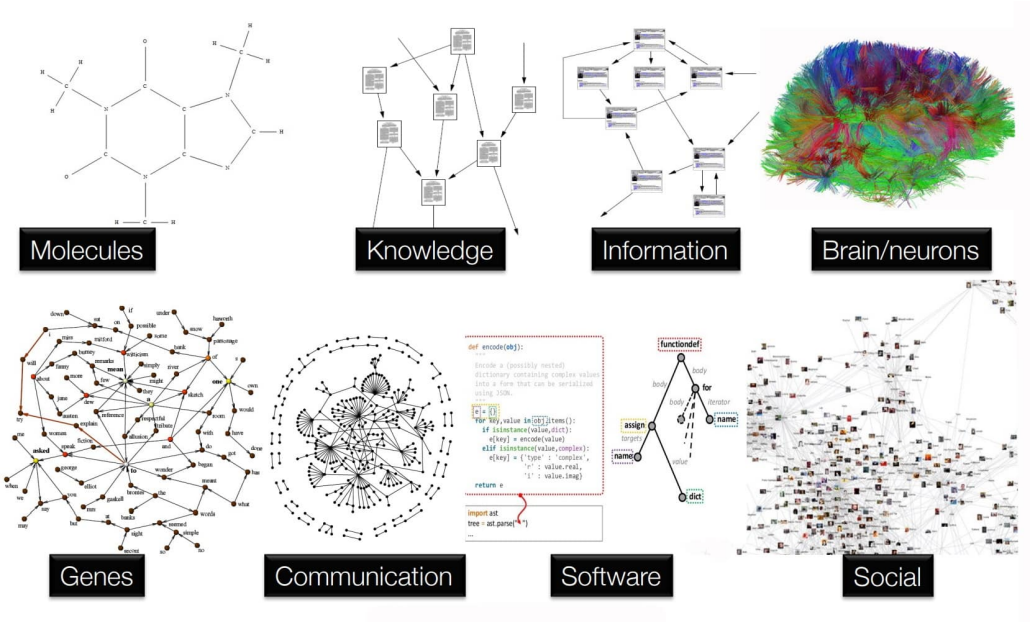


FIGURE 2.1 – Exemple de graphes.
Source: NVIDIA, 2022.

Les GNNs sont une famille de modèles qui permettent de traiter ce type de structures de données. Ces modèles sont majoritairement basés sur des opérations de convolution et d'agrégation, similairement aux opérations de convolution et de pooling utilisées dans les réseaux de neurones pour les modalités plus classique comme les images. On retrouve de même dans les GNNs des architectures avancées, inspirées des réseaux de neurones classiques, comme les réseaux résiduels [16], les réseaux récurrents [31] ou l'attention [59, 7].

Les applications les plus courantes de ces réseaux incluent la classification [27] de documents, la détection de fraudes [36] et les systèmes de recommandation [15]. En revanche, la génération de graphes est moins répandue et se limite souvent dans la littérature à la génération de petites molécules [25, 53].

2.2 Métriques et distances

Il existe de nombreuses distances et métriques spécifiques permettant d'analyser les distributions et les ensembles de données sous forme de nuages de points. Dans cette section, nous présentons certaines de ces mesures, qui sont couramment utilisées dans les articles présentés plus loin.

2.2.1 Kullback-Leibler Divergence (KLD)

Soit deux distributions de probabilités discrètes P et Q sur un ensemble X.

$$D_{\mathrm{KL}}(P\|Q) = \sum_{x\in X} P(x)\log\frac{P(x)}{Q(x)} \tag{2.1}$$

La Divergence de Kullback-Leibler est une mesure de la dissimilarité entre deux distributions de probabilité. Elle évalue la quantité d'information perdue lorsque l'on tente d'approximer une distribution par une autre. La KLD n'est pas une distance, elle ne satisfait pas la propriété de symétrie ($D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$) ni l'inégalité triangulaire ($D_{\mathrm{KL}}(P\|R) \not\leq D_{\mathrm{KL}}(P\|Q) + D_{\mathrm{KL}}(Q\|R)$).

2.2.2 Haussdorf Distance (HD)

Soit X et Y deux nuages de points.

$$d_{\mathrm{HD}}(X,Y) = \max\left\{\sup_{x\in X}\inf_{y\in Y} d(x,y), \sup_{y\in Y}\inf_{x\in X} d(x,y)\right\} \tag{2.2}$$

La distance de Hausdorff est une mesure quantitative utilisée pour évaluer la similarité ou la dissimilarité entre deux ensembles de points dans un espace métrique. Elle calcule la plus grande distance d'un point d'un ensemble à son point le plus proche dans l'autre ensemble. En d'autres termes, elle capture la plus grande distance entre les ensembles, ce qui en fait un outil utile pour comparer des formes, des structures ou des distributions de points.

2.2.3 Chamfer Distance (CD)

Soit X et Y deux nuages de points.

$$d_{\mathrm{CD}}(X,Y) = \sum_{x\in X}\min_{y\in Y}\|x-y\|_2^2 + \sum_{y\in Y}\min_{x\in X}\|x-y\|_2^2 \tag{2.3}$$

La distance de Chamfer est une mesure de similarité entre deux ensembles de points dans un espace métrique. Elle évalue la proximité entre les points des ensembles en calculant la somme des distances entre chaque point d'un ensemble et son point le plus proche dans l'autre ensemble.

2.2.4 Earth Mover Distance (EMD)

Soit X et Y deux nuages de points tels que $|X| = |Y|$, et $\phi : X \to Y$ une bijection.

$$d_{\mathrm{EMD}}(X,Y) = \min_{\phi:X\to Y}\sum_{x\in X}\|x-\phi(x)\|_2 \tag{2.4}$$

La distance du transport optimal, également appelée distance du "Earth Mover", est une mesure de similarité entre deux distributions de masse dans un espace métrique. Elle évalue le coût minimum nécessaire pour déplacer une distribution de masse en une autre en respectant certaines contraintes de déplacement. Cette distance est couramment utilisée pour comparer des distributions de données, telles que des histogrammes, des vecteurs de caractéristiques ou des nuages de points, en prenant en compte non seulement les distances entre les éléments correspondants, mais aussi les coûts associés à leur déplacement.

2.2.5 Jensen-Shannon Divergence (JSD)

Soit S_g un ensemble de nuages de points générés et S_r un ensemble de nuages de points de référence.

$$\text{JSD}(S_g, S_r) = \frac{1}{2} D_{\text{KL}}(S_g \| M) + \frac{1}{2} D_{\text{KL}}(S_r \| M), \quad M = \frac{1}{2}(S_g + S_r) \tag{2.5}$$

La divergence de Jensen-Shannon est une mesure de la similarité entre deux distributions de probabilité. Elle est calculée comme la moyenne des KLDs entre chaque distribution et la moyenne de ces distributions. Contrairement à la KLD, la JSD est symétrique et bornée entre 0 et 1. Cependant, la JSD utilise la distribution globale des nuages de points et non la distribution des nuages de points individuellements. Ainsi, un modèle qui produit toujours une "forme moyenne" peut obtenir un score JSD parfait sans apprendre de distributions significatives.

2.2.6 Coverage (COV)

Soit S_g un ensemble de nuages de points générés, S_r un ensemble de nuages de points de référence et D une distance entre nuages de points.

$$\text{COV}(S_g, S_r) = \frac{|\{\arg\min_{Y \in S_r} D(X, Y) | X \in S_g\}|}{|S_r|} \tag{2.6}$$

La couverture évalue le nombre de nuages de points de référence qui sont appariés à au moins une forme générée. La couverture permet de quantifier la diversité des générations mais est sensible à la perte de modes, cependant elle n'évalue pas la qualité des nuages de points générés. Ainsi, des nuages de points générés de faible qualité mais diversifiés peuvent obtenir des scores de couverture élevés.

2.2.7 Minimum Matching Distance (MMD)

Soit S_g un ensemble de nuages de points générés, S_r un ensemble de nuages de points de référence et D une distance entre nuages de points.

$$\text{MMD}(S_g, S_r) = \frac{1}{|S_r|} \sum_{Y \in S_r} \min_{X \in S_g} D(X, Y) \tag{2.7}$$

La distance de correspondance minimale, est une mesure qui évalue la différence entre deux ensembles ordonnés. Elle calcule la plus petite somme des distances entre les éléments des deux ensembles. Cependant, la MMD est en réalité très peu sensible aux nuages de points de qualité médiocre dans S_g, étant donné leur faible probabilité d'être appariés avec des nuages de points réels dans S_r. Dans un cas extrême, on pourrait envisager que S_g soit composé principalement de nuages de points de très mauvaise qualité, ainsi que de nuages de point pratiquement identiques à ceux de S_r. Dans ce cas, on obtiendrait un score de raisonnablement élevé.

2.2.8 1-Nearest Neighbor Accuracy (1-NNA)

Soit S_g un ensemble de nuages de points générés, S_r un ensemble de nuages de points de référence, N_X les voisins les plus proches de X dans $S_{-X} = S_r \cup S_g - \{X\}$, N_Y les voisins les plus proches de Y dans $S_{-Y} = S_r \cup S_g - \{Y\}$ et $\mathbb{1}$ la fonction indicatrice.

$$\text{1-NNA}(S_g, S_r) = \frac{\sum_{X \in S_g} \mathbb{1}[N_X \in S_g] + \sum_{Y \in S_r} \mathbb{1}[N_Y \in S_r]}{|S_g| + |S_r|} \tag{2.8}$$

Pour chaque point d'un ensemble de nuages de points, cette mesure évalue par une classification par plus proche voisin, si celui-ci provient de S_g ou de S_r. La précision de cette classification est ensuite calculée comme la moyenne des précisions de chaque point de S_g et de S_r.

En supposant que S_g et S_r soient échantillonnés à partir de la même distribution, la précision d'un tel classificateur devrait converger vers 50% avec un nombre suffisant d'échantillons. Plus la précision se rapproche de 50%, plus les similarités entre S_g et S_r sont prononcées, ce qui indique une meilleure capacité du modèle à apprendre la distribution cible. Dans notre contexte, le plus proche voisin peut être calculé à l'aide de la CD ou de l'EMD. Contrairement à la JSD, le 1-NNA considère la similarité entre les distributions de formes plutôt qu'entre les distributions totale des points. Contrairement au COV et au MMD, la 1-NNA mesure directement la similarité des distributions et prend en compte à la fois la diversité et la qualité.

2.3 Modèles génératifs

Les modèles génératifs sont une famille de modèles qui permettent de générer de nouvelles données d'une distribution de données au préalablement apprise. Ces modèles sont très utilisés dans le domaine de l'apprentissage automatique pour générer des images, du texte ou encore de la musique. Ces modèles sont encore relativement peu utilisés dans le domaine de l'ingénierie pour générer des pièces industrielles.

Il existe plusieurs sous familles de modèles génératifs, chacune basées sur des principes différents, possédant ainsi des avantages et des inconvénients. Il est donc important de bien comprendre les différences entre ces modèles pour pouvoir choisir le modèle le plus adapté à la problématique. Plusieurs études [14, 18, 67] ont déjà été réalisées pour comparer ces modèles, nous nous baserons donc partiellement sur celles-ci pour présenter les modèles les plus pertinents pour notre problématique.

2.3.1 Generative Adversarial Network (GAN)

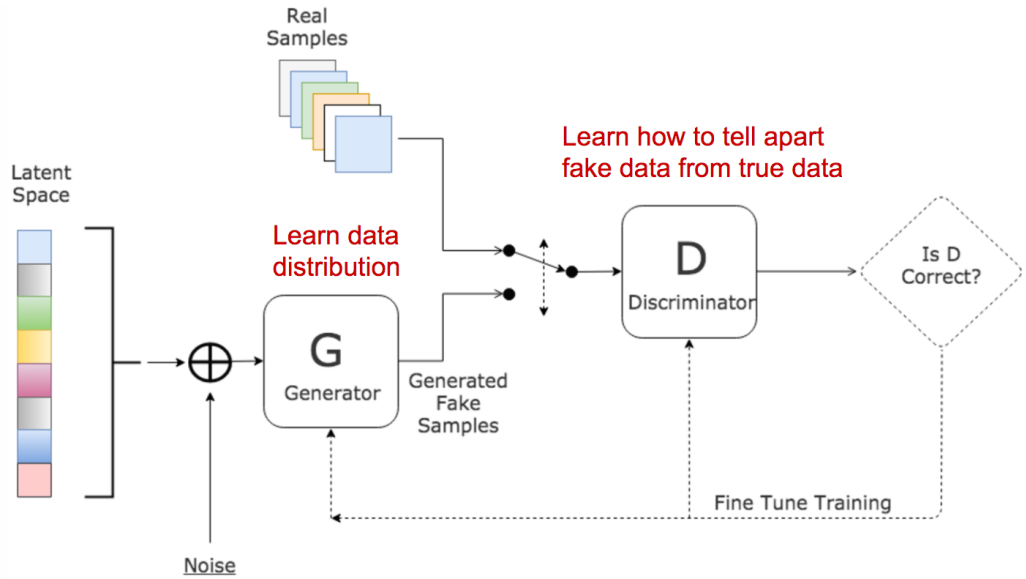


Figure 2.2 – Architecture d'un GAN.
Source: Lilian Weng, 2017.

Les GANs [17] sont la famille de modèles génératifs la plus renommée. Ces modèles reposent sur un principe compétitif impliquant deux réseaux de neurones, visibles sur la figure 2.2. Le premier réseau, connu sous le nom

de générateur, a pour objectif de produire de nouvelles données. Le deuxième réseau, appelé discriminateur, est chargé de distinguer les données générées par le générateur des données réelles. Le générateur est entraîné à tromper le discriminateur tandis que le discriminateur est entraîné à identifier les données générées par rapport aux données réelles. Cette compétition entre les deux réseaux permet de former le générateur à générer des données de plus en plus réalistes. Ce type d'apprentissage est auto-supervisé, car il ne nécessite pas l'utilisation d'annotations sur les données pour entraîner le modèle.

Mathématiquement, on peut poser le problème d'optimisation suivant:

$$\min_{\mathrm{G}} \max_{\mathrm{D}} L(\mathrm{D}, \mathrm{G}) = \mathbb{E}_{x \sim p(\boldsymbol{x})}[\log \mathrm{D}(x)] + \mathbb{E}_{z \sim p(\boldsymbol{z})}[\log(1 - \mathrm{D}(G(z)))] \tag{2.9}$$

Les GANs ont su démontrer leur efficacité pour générer des images réalistes. Cependant, ces modèles sont très difficiles à entraîner [4]. Les GANs sont par exemple sujets à de nombreux problèmes [64], tel que le problème de *mode collapse*, où le générateur génère toujours la même image, mais aussi le problème de *non convergence*, où le générateur et/ou le discriminateur ont une fonction de cout instable et ne convergent ainsi pas vers un équilibre de Nash, ou encore au problème de *vanishing gradient*, où le discriminateur devient trop efficace et empêche le générateur d'apprendre.

Au fil des années, de nombreuses améliorations [51], variations (WGAN [5], etc.) et cas d'applications (CycleGAN [66], SGAN [43], SRGAN [29], DragGAN [45], etc.) ont été proposées, mais ces modèles restent complexes à entraîner et à évaluer. De plus, ces modèles sont très sensibles aux hyperparamètres et nécessitent une grande quantité de données pour être efficaces.

Face à ces inconvénients, et puisque nous ne possédons pas de grandes quantités de données, nous avons choisi de ne pas utiliser cette famille de modèles.

2.3.2 Variational Auto-Encoder (VAE)

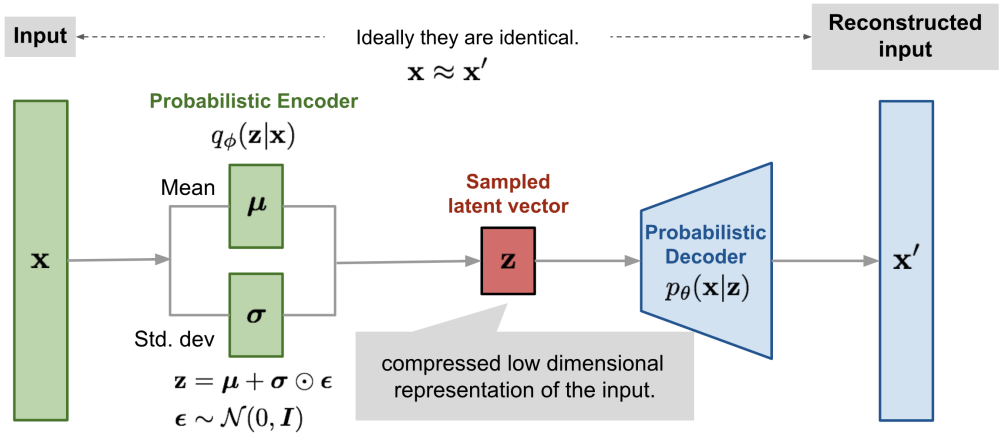


FIGURE 2.3 – Architecture d'un VAE.
Source: Lilian Weng, 2018.

Les VAEs [24, 26, 13] constituent une autre famille de modèles génératifs, également bien connue comme les GANs. Ces modèles reposent sur l'entraînement simultané de deux réseaux de neurones: un encodeur et un décodeur, visibles sur le figure 2.3. L'objectif de l'encodeur est de transformer les données d'entrée en une distribution de probabilité, tandis que le décodeur génère de nouvelles données à partir de cette distribution. Comme pour les GANs, ces modèles visent à estimer une distribution de données qui se rapproche le plus

possible de la distribution des données d'entraînement, c'est-à-dire qu'ils apprennent à reproduire fidèlement les données d'origine.

La particularité inhérente aux VAEs réside dans l'espace latent intermédiaire situé entre l'encodeur et le décodeur. La recherche sur l'interprétabilité des réseaux de neurones et leur visualisations [44] établissent que les espaces latents permettent d'extraire les informations sous-jacentes (non directement perceptibles) des données d'entrée. Travailler sur ces informations s'avère avantageux car elles décrivent plus simplement les données d'entrée. De même, chez les VAEs la dimension de cette espace latent est configurée par l'architecture du réseau et peut être réduite à volonté. L'encodeur et le décodeur peuvent ainsi être conceptualisés comme des opérateurs de compression et de décompression.

Mathématiquement, on peut modéliser ces variables latentes et nos données d'entrée par une distribution jointe $p(\boldsymbol{x}, \boldsymbol{z})$. Les approches génératives ont pour but de trouver un modèle maximisant la vraissemblance de nos données d'entrée $p(\boldsymbol{x})$. En pratique, maximiser directement la vraissemblance est impossible car cela reviendrait à calculer la marginalisation:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} \tag{2.10}$$

Cependant, il est tout de même possible de trouver une borne inférieure de l'évidence, appelée Evidence Lower Bound (ELBO):

$$p(\boldsymbol{x}) \propto \log p(\boldsymbol{x}) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{x})}\right] - D_{\mathrm{KL}}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}|\boldsymbol{x})) \tag{2.11}$$

$$\geq \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{2.12}$$

Ainsi, puisque la KLD est toujours positive et car $\log p(\boldsymbol{x})$ est constant par rapport aux paramètres ϕ de l'encodeur, maximiser l'ELBO (2.12) revient à maximiser l'évidence et donc la vraissemblance de nos données d'entrée $p(\boldsymbol{x})$. En simplifiant l'ELBO, on dérive une fonction de coût pour l'entrainement du VAE:

$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] = \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})]}_{\text{reconstruction term}} - \underbrace{D_{\mathrm{KL}}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}))}_{\text{prior matching term}} \tag{2.13}$$

Une fois la convergence atteinte, l'intéret de cet espace latent, lorsqu'il est accompagné de sont decodeur, est qu'il permet de générer de nouvelles données, par example en échantillonant $z = \mu + \sigma \odot \epsilon$, ou bien en interpolant entre deux points latents, ou encore en modifiant légèrement un point spécifique de cet espace.

Tout comme les GANs, de nombreuses améliorations (β-VAE [8, 20, 2], f-VAE [55]) et variations (SetVAE [23], AutoDecoder [52], GraphVAE [53]) ont été proposées pour les VAEs. Ces modèles sont plus faciles à entraîner que les GANs et présentent une plus grande stabilité. Cependant, les VAEs ont tendance à générer des données floues et peu réalistes [61], et en général produisent des résultats de moins bonne qualité que les GANs, en particulier pour des résolutions élevées.

2.3.3 Normalizing Flow (NF)

Les NFs [28] constituent une autre catégorie de modèles génératifs qui ont suscité un intérêt croissant au cours des dernières années. Cette approche gagne en popularité du fait de sa capacité à opérer directement sur les densités de probabilité, ouvrant ainsi la voie au calcul précis des probabilités d'événements spécifiques. Ces modèles se basent sur des transformations inversibles, bijectives, continues et différentiables. Ces transformations, visibles sur la figure 2.4, sont appliquées à une distribution de base, généralement une distribution

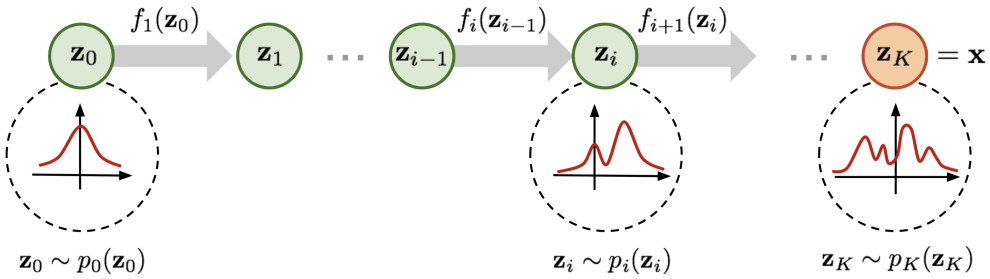
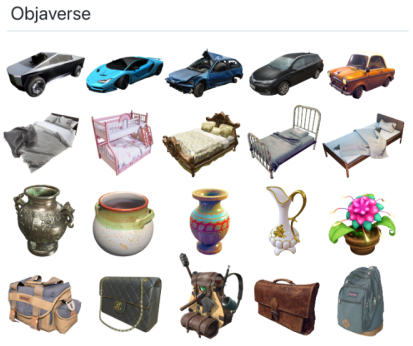


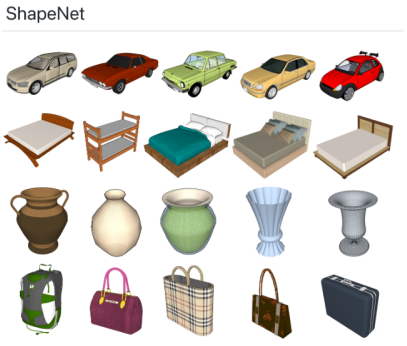
FIGURE 2.4 – Architecture d'un NF.
Source: Lilian Weng, 2018.

simple comme une gaussienne isotropique, pour obtenir une distribution plus complexe et plus proche de la distribution des données réelles.

Les transformations inversibles utilisées dans les NFs sont souvent paramétrisées par des réseaux de neurones, ce qui permet d'apprendre des fonctions non linéaires. En utilisant plusieurs transformations en séquence, on peut construire des modèles génératifs capables de capturer des distributions complexes. En pratique, toutefois, les contraintes inhérentes à ces transformations limitent considérablement les architectures envisageables, restreignant ainsi la capacité de ces moddèles à surpasser les performances génératives d'autres méthodes.

Dans la littérature, l'application de ces réseaux aux types de données qui nous intéressent est relativement limitée. De manière générale, leur utilisation est restreinte pour toutes les applications traitant des données de grande dimension. Une exception notable est représentée par PointFlow [62] qui aura posé certaines bases pour évaluer les réseaux génératifs de nuages de points, notamment via la création d'un dataset de référence qui repose sur ShapeNet [9], modifié par Furthest Point Sampling pour contenir uniquement 2048 points par nuages.





Dataset	# Objects	# Classes
YCB [5]	77	5
BigBIRD [69]	125	–
KIT [35]	219	145
IKEA [43]	219	11
Pix3D [71]	395	9
GSO [19]	1K	17
EGAD [51]	2K	–
PhotoShape [53]	5K	1
ABO [12]	8K	63
3D-Future [22]	10K	34
ShapeNet [7]	51K	55
Objaverse	**818K**	**21K**

FIGURE 2.5 – Exemple de datasets d'objets 3D.

Il existe cependant de nombreux autres datasets 3D, tels que ModelNet[60], KITTI [33], S3DIS [6] ou encore le récent Objaverse [10] et sa version XL [11], visibles sur la figure 2.5.

2.3.4 Variational Diffusion Model (VDM)

Les VDMs [12] sont la famille de réseaux générateurs la plus récente et aussi la plus performante. La manière la plus simple de décrire ces modèles est de les considérer comme une mélange des VAEs et des NFs. En effet, le principe des VDMs est de trouver un processus basé sur des transformation stochastiques, discrètes et

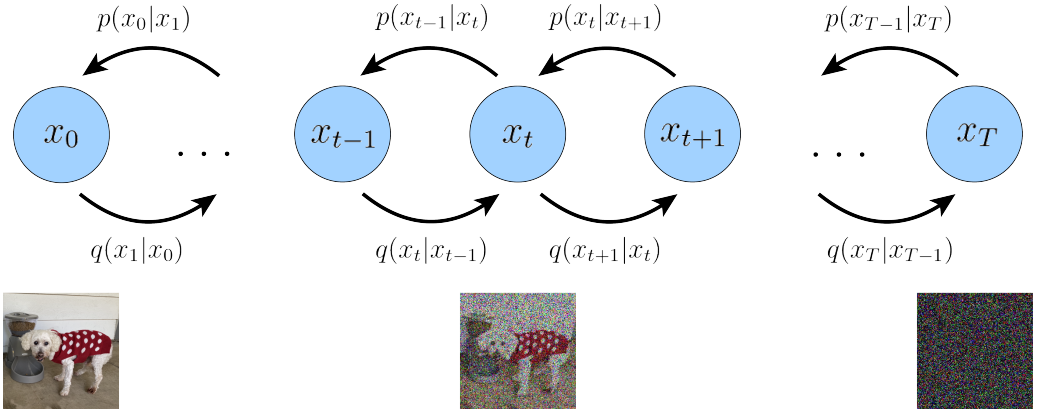


FIGURE 2.6 – Architecture d'un VDM [35].

réversible entre notre distribution de données et une distribution totalement différente, mais que l'on connait de préférence parfaitement mathématiquement.

Plusieurs catégories de modèles sont disponibles pour aborder le problème de la diffusion, parmi lesquelles émerge la plus reconnue, à savoir les Denoising Diffusion Probabilistic Models (DDPMs) [22]. Cette approche vise à identifier une correspondance entre les données observées et une distribution gaussienne standard. Ce processus est appris au moyen d'un modèle paramétrique (i.e. un réseau de neurones).

Dans leur architecture, les VDMs peuvent être vus comme une chaine de Markov de VAEs hiérarchiques avec trois restrictions notable:

- La dimension latente est exactement égale à la dimension des données d'entrée.
- La structure de l'encodeur est fixée et pré-définie. Il s'agit d'un encodeur linéaire gaussien, c'est-à-dire une distribution gaussienne centrée autour de la sortie de l'étape précédente.
- Les paramètres de l'encodeur varient au cours du temps de sorte que la distribution latente à l'étape finale T soit une gaussienne standard.

On note q les "encodeurs" et p les "décodeurs" des VAEs de la chaine de Markov, $x_0 \sim \boldsymbol{x}_0$ un échantillon de notre distribution de données, $x_T \sim \boldsymbol{x}_T$ un échantillon d'une gaussienne isotropique, et $x_t \sim \boldsymbol{x}_t$ tout échantillon intermédiaire, avec t le temps de diffusion, T le temps final de diffusion. On désigne également les transitions de x_t à x_{t+1} comme le "forward process", et les transition de x_t à x_{t-1} comme le "reverse process".

D'après les contraintes précédentes, on peut écrire pour le forward process:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\alpha_t}\boldsymbol{x}_0, (1-\alpha_t)\mathbf{I}) \tag{2.14}$$

avec $\alpha_t \in [0,1]$ qui évolue en t selon une suite décroissante fixée ou apprenable (i.e. via un réseau de neurones).

Cependant puisque toutes ces opérations sont linéaires et gaussiennes, si l'on souhaite obtenir x_t à partir de x_0, au lieu d'appliquer t fois la relation 2.14, on peut simplifier comme suit:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\overline{\alpha}_t}\boldsymbol{x}_0, (1-\overline{\alpha}_t)\mathbf{I}), \quad \overline{\alpha}_t = \prod_{t=1}^{t} \alpha_t \tag{2.15}$$

L'objectif de la diffusion est de trouver une approximation du processus inverse $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) \approx p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ (i.e. via un réseau de neurones, conditionnée sur t). Cependant via une dérivation de l'ELBO et un conditionnement additionnel, on peut montrer que cela revient à minimiser la KLD entre $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x_0})$ et $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$. Ensuite, via une application de la formule de Bayes, on obtient:

$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x_0}) \propto \mathcal{N}(\boldsymbol{x}_{t-1}; \mu_q(\boldsymbol{x}_t, \boldsymbol{x}_0), \Sigma_q(\boldsymbol{x}_t)) \tag{2.16}$$

avec:

$$\mu_q(\boldsymbol{x}_t, \boldsymbol{x}_0) = \frac{\sqrt{\alpha_t}(1-\overline{\alpha}_{t-1})\boldsymbol{x}_t + \sqrt{\overline{\alpha}_{t-1}}(1-\alpha_t)\boldsymbol{x}_0}{1-\overline{\alpha}_t}, \quad \Sigma_q(\boldsymbol{x}_t) = \frac{(1-\alpha_t)(1-\overline{\alpha}_{t-1})}{1-\overline{\alpha}_t} \tag{2.17}$$

On peut finalement simplifier cette expression via une reparamétrisation:

$$\boldsymbol{x}_0 = \frac{\boldsymbol{x}_T - \sqrt{1-\overline{\alpha}_t}\epsilon_0}{\sqrt{\overline{\alpha}_t}} \implies \mu_q(\boldsymbol{x}_t, \boldsymbol{x}_0) = \frac{1}{\sqrt{\alpha_t}}\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}\sqrt{\alpha_t}}\epsilon_0 \tag{2.18}$$

Si l'on réeffectue une dérivation de l'ELBO avec ces nouvelles expressions, on en conclut qu'il suffit de trouver une approximation $\epsilon_\theta(\boldsymbol{x}_t, t) \approx \epsilon_0$. En pratique on utilise un réseau de neurones que l'on entraine à minimiser $\|\epsilon_0 - \epsilon_\theta(\boldsymbol{x}_t, t)\|_2^2$. Une fois cette approximation trouvée, on peut facilement remonter à $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$. Formellement, l'entrainement de ce réseau de neurones est décrit dans l'algorithme 1.

Algorithm 1 DDPM training

Require: $T \in \mathbb{N}^\star$: number of diffusion steps
Require: $\alpha_t \in \mathbb{R}^T$: variance schedule
Require: $\boldsymbol{x}_0$: data distribution to be learned
Require: ϵ_θ: neural network

repeat
  $x_0 \sim \boldsymbol{x_0}$
  $t \sim \text{Uniform}(\llbracket 1, T \rrbracket)$
  $\epsilon \sim \mathcal{N}(0, \mathbf{I})$
  take gradient step on $\nabla_\theta \|\epsilon - \epsilon_\theta(\sqrt{\overline{\alpha}_t}x_0 + \sqrt{1-\overline{\alpha}_t}\epsilon, t)\|_2^2$
until converged
return ϵ_θ

Après avoir achevé l'entraînement adéquat de notre modèle, on peut désormais appliquer $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ itérativement pour passer d'un échantillon x_T à sa prédiction $\hat{x}_0$. Dans les faits, le réseau génère des reconstructions qui ressemblent fortement à nos données d'apprentissage, créant ainsi de nouvelles données. Formellement, l'échantillonnage est décrit dans l'algorithme 2.

Algorithm 2 DDPM sampling

Require: $T \in \mathbb{N}^\star$: number of diffusion steps
Require: $\alpha_t \in \mathbb{R}^T$: variance schedule
Require: ϵ_θ: trained neural network

$x_T \sim \mathcal{N}(0, \mathbf{I})$
for $t = T, ..., 1$ **do**
  $\mu_t = \frac{1}{\sqrt{\alpha}}\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}\sqrt{\alpha_t}}\epsilon_\theta(x_t, t)$
  $\sigma_t = \sqrt{\frac{(1-\alpha_t)(1-\overline{\alpha}_{t-1})}{1-\overline{\alpha}_t}}$
  $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ if $t > 1$, else $\epsilon = 0$
  $x_{t-1} = \mu_t + \sigma_t\epsilon$
end for
return x_0

Il est possible de démontrer théoriquement [35] l'équivalence entre les VDMs et les méthodes de score matching [54] lorsque T tend vers l'infini. Les méthodes de score matching, constituent une famille de techniques permettant l'estimation de la densité de probabilité associée à un ensemble de données. Elles se

basent exclusivement sur le calcul du gradient de cette densité de probabilité, éliminant ainsi la nécessité du calcul laborieux d'une constante de normalisation. Une fois le gradient estimé (e.g. via un réseau de neurones), la densité de probabilité peut être retrouvée au moyen de méthodes d'échantillonnage telles que la méthode du recuit de Langevin [54].

Latent Diffusion Model (LDM)

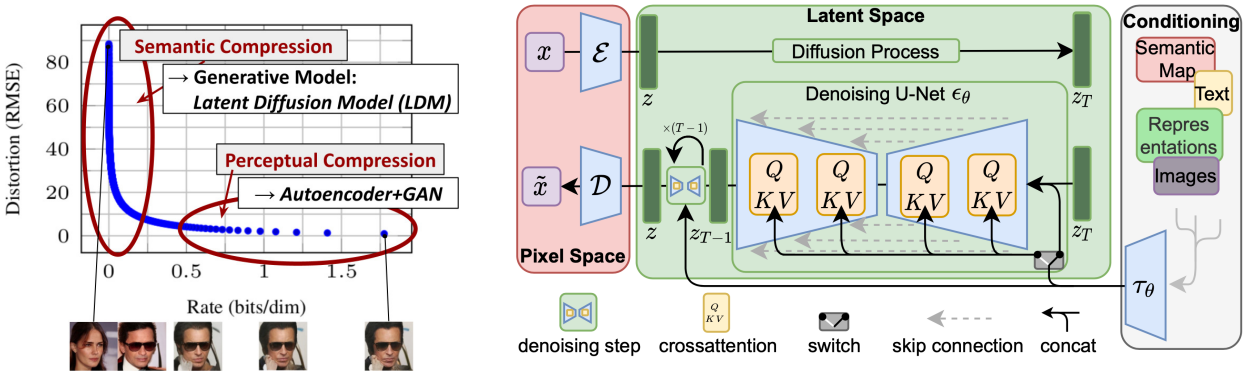


FIGURE 2.7 – Architecture d'un LDM [50].

Une amélioration significative des VDMs réside dans la mise en œuvre intelligente des espaces latents. Cette méthode, dénommée LDM [50], repose sur l'observation selon laquelle l'exploitation des informations latentes (souvent de dimensions nettement réduites), tout comme pour les VAEs, confère des avantages substantiels en termes de représentativité des données. La transition des VDMs vers les LDMs consiste en l'introduction préalable d'un second modèle, qu'il soit paramétrique ou non, destiné à obtenir une représentation alternative de nos données.

Les VAEs sont fréquemment employés à cet effet. L'entrainement des LDMs ne changent pas par rapport aux VDMs, seul le domaine d'apprentissage du modèle est modifié. Ainsi cette approche induit généralement une réduction de la complexité du réseau, entraînant ainsi une diminution du temps nécessaire à l'entraînement, tout en exerçant une influence forte sur la qualité des résultats obtenus.

Conditionnement & Guidance

Jusqu'à présent, les modèles génératifs ont démontré leur capacité à générer des données conformes à une distribution de données apprise. Cependant, il est fréquemment nécessaire d'échantillonner une sous-distribution spécifique de nos données. À cette fin, il est possible de conditionner le modèle en utilisant une seconde donnée d'entrée, tel qu'une image, un scalaire, un audio, ou du texte, à l'instar des modèles de type Text to Image (T2I) tels que DALL·E 2 ou Stable Diffusion.

Deux méthodes existent actuellement pour permettre à des modèles de diffusion de générer des données conditionnées.

La première méthode est appelée Classifier Guidance [12] et repose sur l'entrainement d'un classificateur annexe $f_\phi(y|x_t)$ entrainé sur des données bruitées x_t ainsi que leur classes associées y. Le logarithme du gradient de ce classificateur par rapport à x_t est ensuite utilisé pour guider le modèle de diffusion à générer

des données de classe y. Ainsi selon la formulation du score matching on obtient:

$$\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t|y) = \nabla_{\boldsymbol{x}_t} \log p\left(\frac{p(\boldsymbol{x}_t)p(y|\boldsymbol{x}_t)}{p(y)}\right) \tag{2.19}$$

$$= \underbrace{\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)}_{\text{unconditional score}} + \underbrace{\nabla_{\boldsymbol{x}_t} \log q(y|\boldsymbol{x}_t)}_{\text{adversarial gradient}} \tag{2.20}$$

Si l'on remplace $q(y|\boldsymbol{x}_t)$ par $f_\phi(y|\boldsymbol{x}_t)$ et que l'on introduit un facteur de guidage $\gamma \in \mathbb{R}$, on obtient:

$$\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t|y) \approx \nabla_{\boldsymbol{x}_t} \log q(\boldsymbol{x}_t) + \gamma \nabla_{\boldsymbol{x}_t} \log f_\phi(y|\boldsymbol{x}_t) \tag{2.21}$$

$$= -\frac{1}{\sqrt{1-\overline{\alpha}_t}} \epsilon_\theta(\boldsymbol{x}_t, t) + \gamma \nabla_{\boldsymbol{x}_t} \log f_\phi(y|\boldsymbol{x}_t) \tag{2.22}$$

$$= -\frac{1}{\sqrt{1-\overline{\alpha}_t}} \left[\epsilon_\theta(\boldsymbol{x}_t, t) - \gamma\sqrt{1-\overline{\alpha}_t} \nabla_{\boldsymbol{x}_t} \log f_\phi(y|\boldsymbol{x}_t)\right] \tag{2.23}$$

On identifie alors:

$$\epsilon_\theta(\boldsymbol{x}_t, t, c) = \epsilon_\theta(\boldsymbol{x}_t, t) - \gamma\sqrt{1-\overline{\alpha}_t} \nabla_{\boldsymbol{x}_t} \log f_\phi(y|\boldsymbol{x}_t) \tag{2.24}$$

FIGURE 2.8 – Gradients de $f_\phi(y|\boldsymbol{x}_t)$ de la Classifier Guidance (CG).
Source: Paweł Pierzchlewicz, 2023.

La seconde méthode est appelée Classifer-Free Guidance [21, 41] et repose sur l'entraînement d'un unique réseau de neurones ayant pour objectif d'apprend à la fois la distribution conditionnelle et non conditionnelle. En réarrangeant l'équation 2.20, on obtient:

$$\nabla_{\boldsymbol{x}_t} \log p(y|\boldsymbol{x}_t) = \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t, y) - \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t) \tag{2.25}$$

En subsituant l'équation 2.21 dans l'équation 2.25, on obtient:

$$\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t|y) = \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t) + \gamma(\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t|y) - \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)) \tag{2.26}$$

$$= \underbrace{\gamma \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t|y)}_{\text{conditional score}} + \underbrace{(1-\gamma) \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)}_{\text{unconditional score}} \tag{2.27}$$

On en déduit de 2.27, que la distribution conditionnelle est une interpolation linéaire entre la distribution conditionnelle et non conditionnelle. Ainsi, en utilisant un facteur de guidage γ, on peut contrôler la contribution de la distribution conditionnelle dans la distribution finale. Si $\gamma = 0$, alors la distribution conditionnelle est ignorée, et si $\gamma = 1$, alors la distribution conditionnelle est utilisée.

Cette approche présente divers avantages:

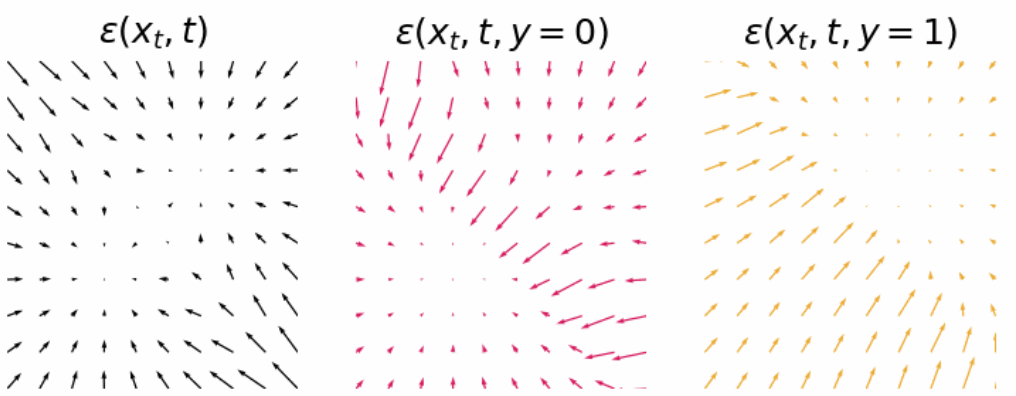


FIGURE 2.9 – Gradients conditionnés et non conditionnés via la Classifier-Free Guidance (CFG). Source: Paweł Pierzchlewicz, 2023.

— Elle s'appuie sur un unique réseau de neurones, contrairement à la méthode guidée par classificateur qui en utilise deux.
— L'entraînement n'est pas excessivement plus complexe; on utilise un entraînement conjoint, car il est extrêmement simple à mettre en œuvre et n'alourdit pas le processus d'entraînement. Ainsi, on procède à un entraînement conditionnel tout en omettant parfois les embeddings de classe afin de réaliser conjointement un entraînement non conditionnel.
— Les données telles que le texte se prêtent difficilement à une classification en classes, et cette approche permet l'utilisation de vecteurs scalaires pour le conditionnement.

Formellement, l'algorithme d'entraînement par CFG est décrit dans l'algorithme 3, l'agorithme d'échantillonnage est décrit dans l'algorithme 4.

Algorithm 3 DDPM training with CFG

Require: $T \in \mathbb{N}^\star$: number of diffusion steps
Require: $\alpha_t \in \mathbb{R}^T$: variance schedule
Require: $p_{\text{uncond}} \in [0, 1]$: probability of unconditional training
Require: $\boldsymbol{x}_0$: data distribution to be learned
Require: $\boldsymbol{c}$: embedding distribution to be learned
Require: ϵ_θ: neural network
1: **repeat**
2: $(x_0, c) \sim (\boldsymbol{x_0}, \boldsymbol{c})$
3: $c \leftarrow \varnothing$ with probability p_{uncond}
4: $t \sim \text{Uniform}(\llbracket 1, T \rrbracket)$
5: $\epsilon \sim \mathcal{N}(0, \mathbf{I})$
6: take gradient step on $\nabla_\theta \|\epsilon - \epsilon_\theta(\sqrt{\overline{\alpha}_t} x_0 + \sqrt{1 - \overline{\alpha}_t}\epsilon, t, c)\|_2^2$
7: **until** converged
8: **return** ϵ_θ

Dans notre cas d'application, nous pouvons conditionner sur les scalaires représentant les performances de nos maillages.

Algorithm 4 DDPM sampling with CFG

Require: $T \in \mathbb{N}^\star$: number of diffusion steps
Require: $\gamma \in \mathbb{R}$: guidance factor
Require: $\alpha_t \in \mathbb{R}^T$: variance schedule
Require: c: class embeddings
Require: ϵ_θ: trained neural network

1: $x_T \sim \mathcal{N}(0, \mathbf{I})$
2: **for** $t = T, ..., 1$ **do**
3: $\quad \epsilon_\theta = \gamma\epsilon_\theta(x_t, t, c) + (1-\gamma)\epsilon_\theta(x_t, t, \varnothing)$
4: $\quad \mu_t = \dfrac{1}{\sqrt{\alpha_t}}\boldsymbol{x}_t - \dfrac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}\sqrt{\alpha_t}}\epsilon_\theta$
5: $\quad \sigma_t = \sqrt{\dfrac{(1-\alpha_t)(1-\overline{\alpha}_{t-1})}{1-\overline{\alpha}_t}}$
6: $\quad \epsilon \sim \mathcal{N}(0, \mathbf{I})$ if $t > 1$, else $\epsilon = 0$
7: $\quad x_{t-1} = \mu_t + \sigma_t\epsilon$
8: **end for**
9: **return** x_0

2.3.5 Auto-Regressive Model (ARM)

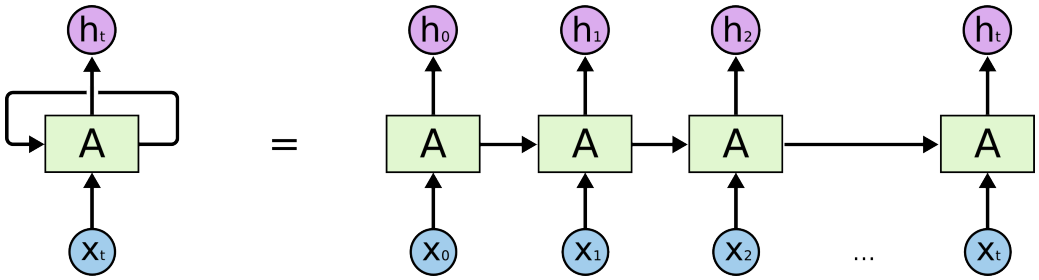


FIGURE 2.10 – Architecture d'un ARM.
Source: Christopher Olah, 2015.

Les modèles auto-régressifs sont des méthodes de génération de séquences qui utilisent les éléments précédents pour prédire chaque élément suivant. Ces modèles sont largement utilisés dans le domaine du traitement du langage naturel, où ils ont montré d'excellentes performances. Cependant, l'application de ces modèles à la génération de graphes présente des défis particuliers en raison de la structure complexe des graphes. En effet, les graphes sont des structures de données non régulières et non séquentielles, ce qui rend difficile l'utilisation des modèles auto-régressifs. Malgré cela, plusieurs approches [40, 32] ont été proposées pour adapter ces modèles à la génération de graphes. Cependant, il est important de noter que ces modèles deviennent de moins en moins précis de manière exponentielle à mesure que la taille de la séquence à générer augmente. De ce fait nous n'avons pas encore utilisé ces modèles dans nos travaux.

2.3.6 Neural Radiance Field (NeRF)

Les NeRFs [37] représentent une autre famille de modèles génératifs qui ont gagné en popularité récemment. Ces modèles ont la capacité de générer des rendus 3D hautement réalistes à partir de données d'entraînement en utilisant des réseaux de neurones. Contrairement aux approches traditionnelles de rendu 3D basées sur des maillages, les NeRFs exploitent des représentations continues et implicites des scènes en décrivant les

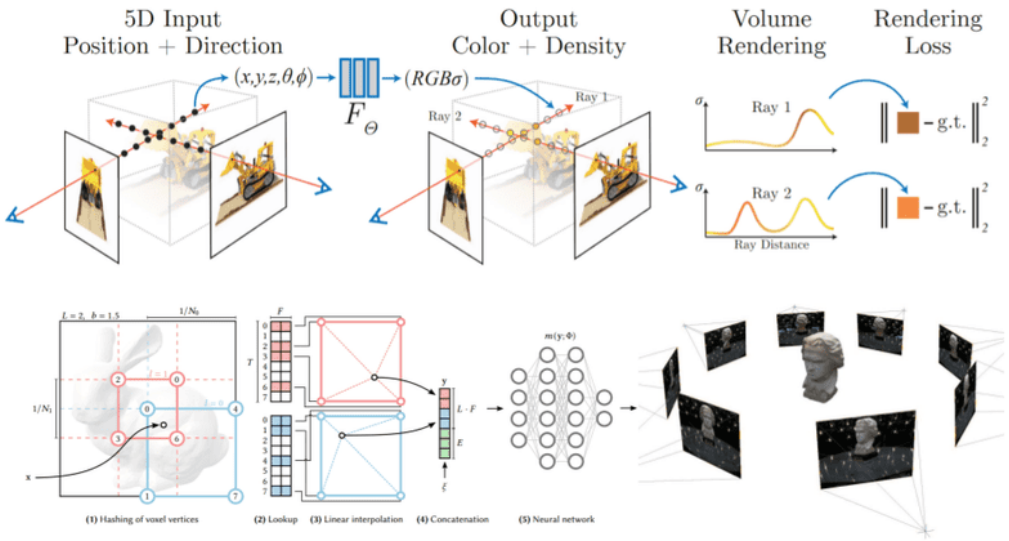


FIGURE 2.11 – Architecture d'un NeRF.
Source: AI Summer, 2022.

propriétés radiométriques et géométriques en chaque point de l'espace 3D.

Le principe des NeRFs est de modéliser une fonction de densité de rayon (ou "ray density function") qui caractérise l'intéraction de la lumière avec les objets de la scène. Cette fonction est ensuite utilisée pour estimer la couleur et la profondeur des rayons traversant la scène, permettant ainsi de générer des images photoréalistes.

Les NeRFs sont donc une alternative aux méthodes traditionnelles de reconstructions de scènes par résolution du problème inverse 3D. Cependant ces modèles peuvent aussi être utilisé conjointement avec d'autres réseaux pour permettre d'obtenir des réseaux génératifs [42, 57, 39].

Dans notre cas, notre jeu de données n'est pas adapté à l'application des NeRFs, car cela nécessiterait un processus de prétraitement complexe de notre ensemble de données (comprenant la conversion de nos maillages/scènes en images via un moteur de rendu) ainsi qu'un post-traitement conséquent de nos résultats (utilisation du "marching cube" pour passer d'une représentation implicite à un maillage). Par conséquent, nous ne choisirons pas d'adopter cette approche. De plus, dans le contexte industriel, les outils destinés à la manipulation d'objets implicites ne sont pas encore suffisamment avancés pour être déployés en production.

Chapitre 3

Déroulement du stage

Ce chapitre présente un aperçu détaillé du déroulement de mon stage de 6 mois (Du 21 Mars 2023 au 21 Septembre 2023) au sein de Safran. Tout au long de cette période, j'ai travaillé en tant que Stagiaire Ingénieur de Recherche en Machine Learning au sein du département Safran Tech, dans l'équipe physics inFormed machine Learning and numerical EXploration (FLEX) de l'unité de recherche Mathematics Applied to Design and Simulation (MADS) dans le département Digital Sciences and Technologies (DST), dont le but est de développer des outils de simulation et de modélisation pour les besoins de Safran. J'ai été encadré par Xavier Roynard, Michele Alessandro Bucci et Brian Staber.

Je décrirai dans les prochaines sections les différentes étapes de mon stage, les tâches qui m'ont été confiées ainsi que les projets auxquels j'ai contribué.

3.1 Lecture de la littérature

Les premiers jours de mon stage ont été dédiés à mon intégration au sein de l'entreprise. J'ai rencontré mes tuteurs de stage qui m'ont présenté l'équipe et les différents membres du département. Une visite des locaux de l'entreprise m'a été proposée, accompagnée d'explications sur les mesures de sécurité en vigueur. J'ai également pris connaissance des outils et des logiciels utilisés dans le cadre de mon projet. Ces premiers jours ont été l'occasion pour moi de participer à des réunions d'équipe, en présence d'autres stagiaires et d'ingénieurs, afin de me familiariser avec les différents projets en cours et de préciser les objectifs de mon stage.

Les deux premières semaines de mon stage ont été dédiées à la lecture approfondie de la littérature scientifique liée à mon domaine d'étude. J'ai effectué des recherches bibliographiques afin de recueillir des informations pertinentes sur les avancées récentes, les théories et les techniques utilisées dans le domaine des modèles génératifs. J'ai majoritairement consulté des articles de conférence et des documents en ligne pour obtenir une vue d'ensemble complète des travaux antérieurs réalisés par des chercheurs et des ingénieurs. Pour appronfondir mes recherches, j'ai également utilisé des outils, tels que Semantic Scholar ou Papers with code, pour trouver les codes sources des papiers ainsi que d'autres papiers ayant pour citation ou référence les articles que j'avais déjà lus, me permettant ainsi de découvrir de nouvelles publications pertinentes.

Lors de ma lecture, j'ai pris des notes (via les logiciels Logseq et Zotero) sur les concepts clés, les méthodologies et les résultats des études. J'ai analysé et comparé les différentes approches proposées dans la littérature afin de mieux comprendre les avantages et les limites de chaque méthode. Cette phase de lecture m'a permis d'acquérir une solide base de connaissances et de me familiariser avec les travaux existants dans le domaine. Ces connaissances préliminaires ont été essentielles pour orienter mes travaux ultérieurs de développement et de recherche lors du stage.

Au cours de cette période, j'ai également eu des discussions régulières avec mes tuteurs de stage pour discuter des articles lus, clarifier certains points et définir la direction à suivre pour mon projet. Ces échanges m'ont permis d'approfondir ma compréhension et de cibler les aspects spécifiques sur lesquels je devais me concentrer lors des prochaines phases de mon stage.

3.2 Prise en main des données

En parallèle de ma lecture de la littérature, j'ai entamé l'exploration des données fournies par Safran. J'ai acquis une compréhension des différents formats de données spécifiques utilisés par l'entreprise pour stocker les résultats des simulations numériques de CFD. De plus, j'ai appris à manipuler ces données en utilisant des outils tels que Paraview [1].

Le principal ensemble de données sur lequel j'ai travaillé pendant mon stage s'appelle Rotor37_1200 [38]. Il s'agit d'un ensemble de données de simulation CFD d'une des 37 aubes du compresseur d'un moteur d'avion. Cet ensemble de données contient 1200 échantillons, qui ont été créé via un processus d'optimisation consistant en l'exploration de paramètres en quête de la maximisation d'un critère de performance, visible sur la figure 3.1.

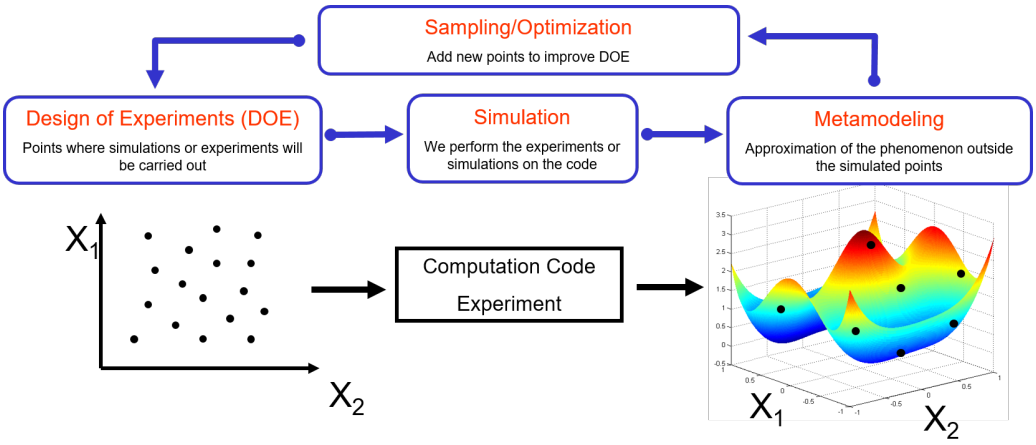


FIGURE 3.1 – Processus d'optimisation ayant permis de générer l'ensemble de données Rotor37_1200.

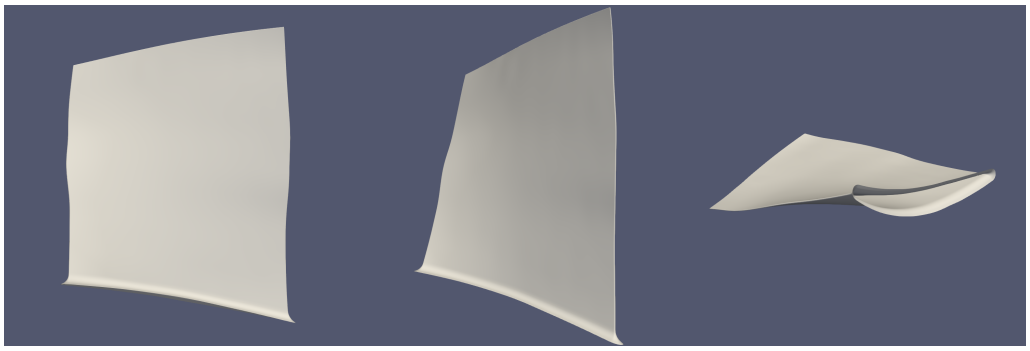


FIGURE 3.2 – Échantillon de l'ensemble de données Rotor37_1200 sous plusieurs angles.

Chaque aube du jeu de données est une déformation d'une aube nominale. Ainsi tous les maillages possèdent le même nombre de nœuds et la même connectivité. Pour donner un ordre de grandeur, chaque maillage est constitué de 29773 nœuds, 59328 triangles et 89100 arêtes.

Chaque échantillon est constitué de deux fichiers distincts. Le premier est un fichier au format .vtk qui contient

le maillage de l'aube, comprenant les positions 3D, les normales et la connectivité de chaque point du maillage. Ce fichier .vtk inclut également les champs physiques associés à chaque point, tels que la température, la pression, etc. Le second fichier est un fichier .csv contenant des métadonnées globales spécifiques à l'échantillon, telles que les entrées et les sorties de la simulation CFD. L'ensemble de ces fichiers pèsent environ 60 Gigaoctet (Go) et sont stockés dans un dossier spécifique read-only sur le cluster de calcul de Safran.

Cet ensemble de données peut être séparé en deux sous-ensembles: un ensemble d'apprentissage de 1000 échantillons (83% des données) et un ensemble de validation de 200 échantillons (17% des données). L'ensemble d'apprentissage est utilisé pour entraîner les modèles génératifs, tandis que l'ensemble de validation est utilisé pour évaluer les performances des modèles génératifs.

Afin de simplifier le chargement des données, j'ai choisi d'utiliser la bibliothèque HuggingFace Datasets [30]. Cette bibliothèque se distingue par son utilisation innovante de la technologie Apache Arrow [49] pour stocker les données de manière tabulaire en colonnes, permettant des lectures sans copie (zero copy reads) ainsi que des opérations de mapping efficaces prenant en charge le multi-threading. Grâce à une approche de chargement paresseux (lazy loading), elle évite les problèmes de mémoire et assure la reproductibilité en sauvegardant en cache les transformations intermédiaires des données.

En complément de Rotor37_1200, j'ai également effectué des travaux sur un ensemble de données plus étendu, baptisé Rotor37_11000, qui comprend 11000 échantillons. La création de cet ensemble de données a suivi le même processus d'optimisation que celui de Rotor37_1200, selon la même aube nominale. Cependant, il est important de noter que les déformations présentes dans Rotor37_11000 sont d'un ordre de grandeur supérieur.

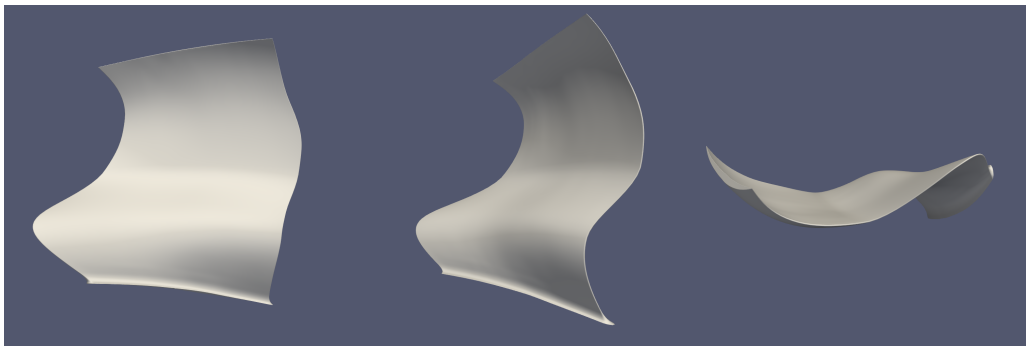


FIGURE 3.3 – Échantillon de l'ensemble de données Rotor37_11000 sous plusieurs angles.

3.3 Description de l'environnement de travail

L'équipe de mes tuteurs est basée à Châteaufort, sur le plateau de Saclay, où se trouve le site de l'entreprise. J'ai réussi à trouver un logement dans le nord de Palaiseau, à environ 40 minutes de trajet en bus. En moyenne, le nombre d'employés présents sur le site s'élève à environ 1200 personnes.

Les locaux de l'entreprise se présentent sous la forme de vastes openspaces, partagés par un maximum d'une dizaine de personnes. Ils sont séparés par de grandes baies vitrées et répartis dans 3 bâtiments sur plusieurs étages. Les bureaux sont spacieux, équipés d'au moins un grand écran, d'un clavier et d'une souris. Nous disposons également de salles de réunion, de salles de détente et d'une salle de sport.

Chaque employé dispose d'une station de travail sous la forme d'un ordinateur portable, connecté à un dock sur le bureau. Afin de réaliser des calculs intensifs, nous avons la possibilité de nous connecter au cluster de calcul local, appelé Rosetta, utilisant le système slurm. Ce cluster est composé d'environ 3000 cœurs CPU, 50 GPU et dispose de plusieurs téraoctets de RAM et de stockage disque. Pour le développement de nos projets, nous

exploitons la forge interne de Safran, qui est une plateforme GitLab dédiée. En outre, chaque employé a accès à la suite professionnelle Office 365, qui facilite la gestion des documents et des e-mails. Pour communiquer, pour les visioconférences nous utilisons principalement Microsoft Teams, qui permet de passer des appels audio et vidéo, de partager son écran et de discuter par écrit, pour les échanges rapides nous utilisons la messagerie instantanée chiffrée de bout en bout Citadel.

La stack technique utilisée par l'équipe est basée sur Python, avec des bibliothèques telles que PyTorch, PyTorch Geometric, PyTorch Lightning, NumPy, SciPy, GPy, Matplotlib, Seaborn, Scikit-Learn, etc. Nous utilisons également des outils de gestion d'environnement virtuels Python tels que Conda et Micromamba, ainsi que de l'outils de versionnement git. Le cluster de calcul étant coupé d'internet, nous utilisons un artifcatory pour télécharger les dépendances nécessaire à nos projets.

3.4 Application de l'état de l'art

À la suite de ma recherche bibliographique, j'ai consacré du temps à expérimenter diverses implémentations des articles que j'avais identifiés, ainsi qu'à travailler sur mon implémentation finale. Voici la liste des implémentations que j'ai évaluées, les idées que j'ai explorées, mes observations relatives à chaque approche, ainsi que mon résultat final, dans un ordre approximativement chronologique.

3.4.1 Test de GraphVAE

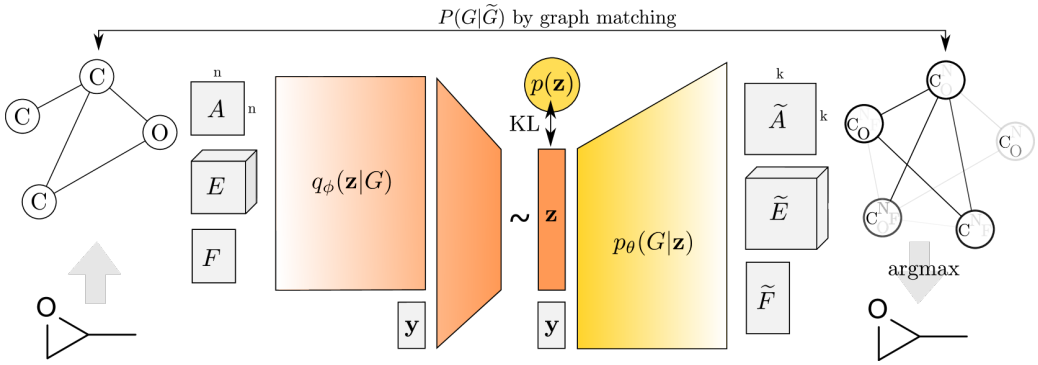


FIGURE 3.4 – Architecture de GraphVAE.

L'une de nos premières initiatives a été de tester des réseaux basés sur les VAEs. Après avoir lu des articles de recherche sur les VAEs, j'ai réalisé plusieurs implémentations sur des images pour me refamiliariser avec ces concepts. Nous avons ensuite étendu ces expérimentations à des architectures spécifiques aux graphes via GraphVAE [53]. Les résultats obtenus étaient encourageants, le réseau était capable de générer des structures, mais la qualité des générations n'était pas exceptionnelle comme visible sur la figure 3.5.

En effet, les générations produites étaient globalement correctes, cependant, elles présentaient d'importantes anomalies dans les régions où la densité de points était élevée. De manière similaire, le réseau présentait une forte tendance au surapprentissage, manifestant une rapide convergence vers des générations excessivement similaires les unes aux autres. De plus, le dimensionnement du réseau (environ 4 millions de paramètres) était disproportionné par rapport à son objectif fonctionnel.

En effet, dans le contexte des graphes, les opérations de "upsampling" n'existent pas de manière unique. Par conséquent, nous avons rencontré des difficultés lors du passage du vecteur latent (représentant une distribution gaussienne) à une représentation sous forme de graphe (noeuds + connectivité) dans le décodeur

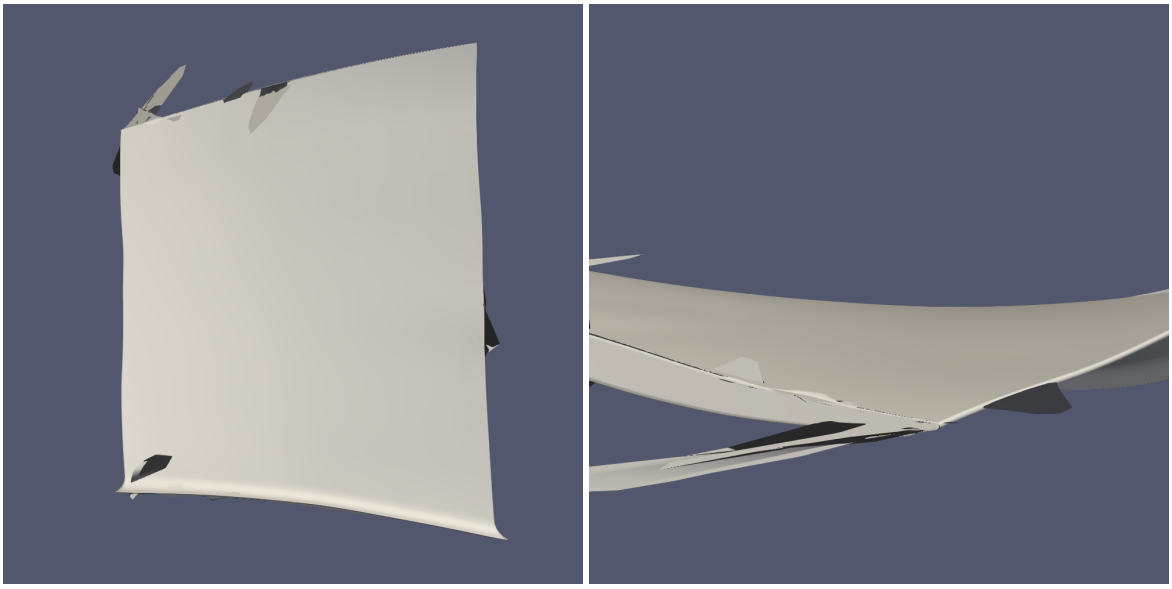


FIGURE 3.5 – Résultats de GraphVAE sur Rotor37_1200.

du VAE.

Une première solution simple consistait en l'utilisation de un ou plusieurs Multi-Layer Perceptrons (MLPs) pour convertir le vecteur latent en un couple de matrices décrivant les positions et la connectivité des noeuds. Cependant, cette approche posait problème en raison de la taille des graphes que nous manipulions. En effet, avec des graphes de $n = 3{,}0 \times 10^4$ nœuds, cela impliquait une matrice de connectivité de taille $n^2 = 9{,}0 \times 10^8$, ce qui faisait aisaiment exploser la complexité lorsque nous utilisions plusieurs MLPs.

Pour donner un ordre de grandeur, si l'on utilisait un espace latent de taille 8, rien que pour prédire les positions 3D des points dans notre maillage (sans prendre en compte la connectivité), l'utilisation d'une seule couche dense impliquait déjà $7{,}2 \times 10^6$ paramètres. Prédire la connectivité était tout simplement impossible, car il aurait fallu une couche dense avec plus de $7{,}2 \times 10^9$ paramètres, ce qui dépassait largement les capacités de calcul de nos ressources GPU.

Une seconde solution consitait à utiliser une architecture plus intelligente, telle que Graph U-Net [16], visible dans la figure 3.6. Cette approche permettait d'éviter l'utilisation de couches denses dans le décodeur grâce à des connexions résiduelles (skip connections). Cependant, ce faisant l'information ne passait pas entièrement par l'espace latent entre le décodeur et l'encodeur. Par conséquent, il était impossible de créer un modèle génératif complet avec cette architecture, puisqu'une partie de l'information pour générer des échantillons était compris dans les skip connections.

Face aux difficultés rencontrées avec les réseaux basés sur les VAE et les limitations de l'architecture Graph U-Net, nous avons pris la décision de mettre de côté ces approches. Et plus largement puisque la connectivité de nos graphes est "locale" (les noeuds des nos maillages sont connectés à leurs voisins proches dans l'espace), nous avons décidé de nous orienter vers des approches basées uniquement sur les positions des noeuds. En effet, la connectivité d'un nuage de points peut être retrouvée via diverses techniques [46, 56, 3].

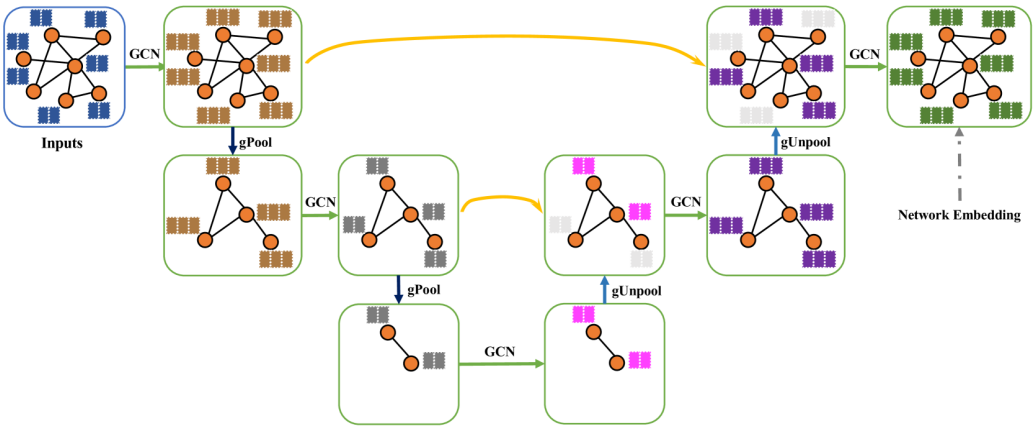


FIGURE 3.6 – Architecture de Graph U-Net.

3.4.2 Présentation de PointNet

Dans le contexte de l'apprentissage sur les nuages de points, une architecture standard est PointNet [47]. PointNet est une architecture basée sur des shared MLPs (pouvant être envisagés comme des convolutions à noyau 1x1), qui permettent de traiter des nuages de points, indépendemment du nombre de points. Cette architecture présente un intérêt notable du fait de son invariance par permutation, ainsi que de sa résilience face à certaines transformations telles que les rotations et les translations.

Par la suite, une avancée significative du modèle PointNet a conduit à l'émergence de PointNet++ [48], qui est désormais reconnu comme l'architecture de référence dans ce domaine. PointNet++ propose une approche hiérarchique en profondeur qui combine judicieusement des techniques d'échantillonnage (telles que le Furthest Point Sampling) et d'agrégation (comme la recherche des K Nearest Neighbors) appliquées aux points du nuage. Cette approche vise à étendre la portée des opérations locales de manière à englober des champs réceptifs globaux, contribuant ainsi à l'amélioration des performances du réseau.

Ces deux architectures sont illustrées sur la figure 3.7.

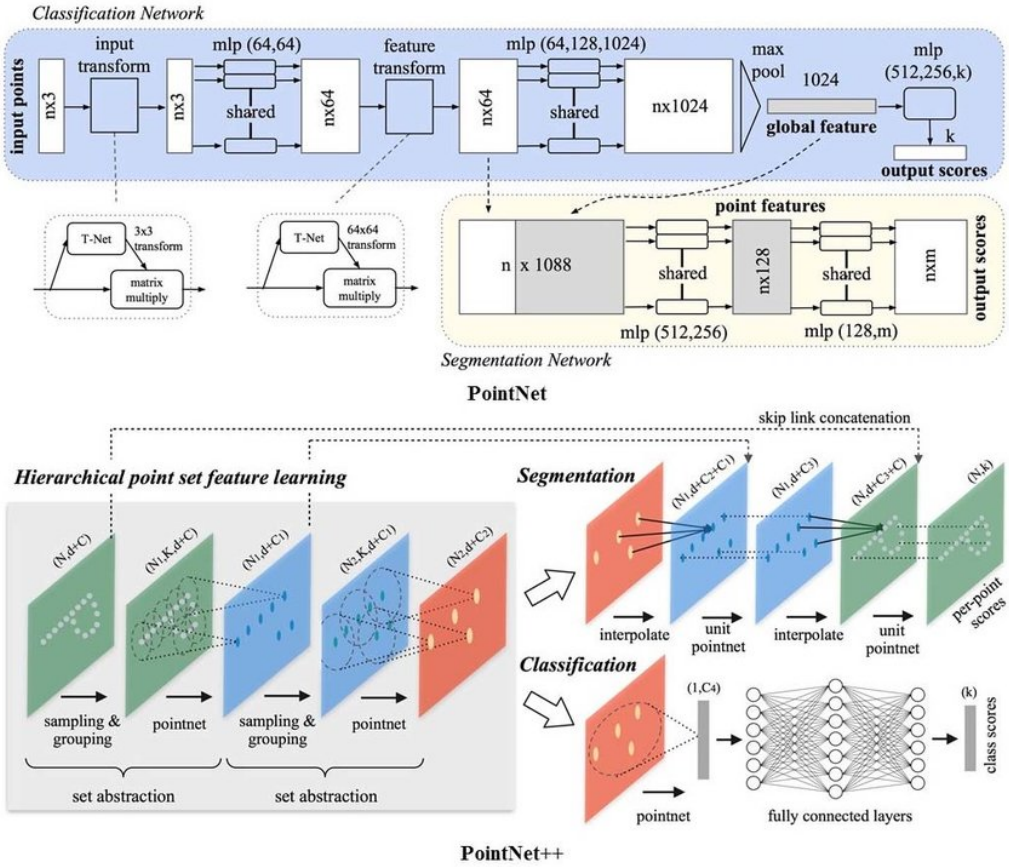


FIGURE 3.7 – Architecture de PointNet et PointNet++.

3.4.3 Présentation des Kernel Point Convolutions (KPConvs)

Une autre architecture largement reconnue dans le domaine du traitement de nuages de points, présentant des similitudes avec PointNet++, est nommée Kernel Point Convolution (KPConv) [58]. Cette architecture utilise un échantillonnage basé sur une grille de voxels et des opérations d'agrégation via des KPConvs, qui sont des convolutions sur boules. Les auteurs de KPConv proposent deux architectures, Kernel Point Convolutional Neural Network (KP-CNN) permettant de traiter des problèmes de classification, et Kernel Point Fully Convolutional Neural Network (KP-FCNN) permettant de traiter des problèmes de segmentation. Ces deux architectures sont illustrées sur la figure 3.8.

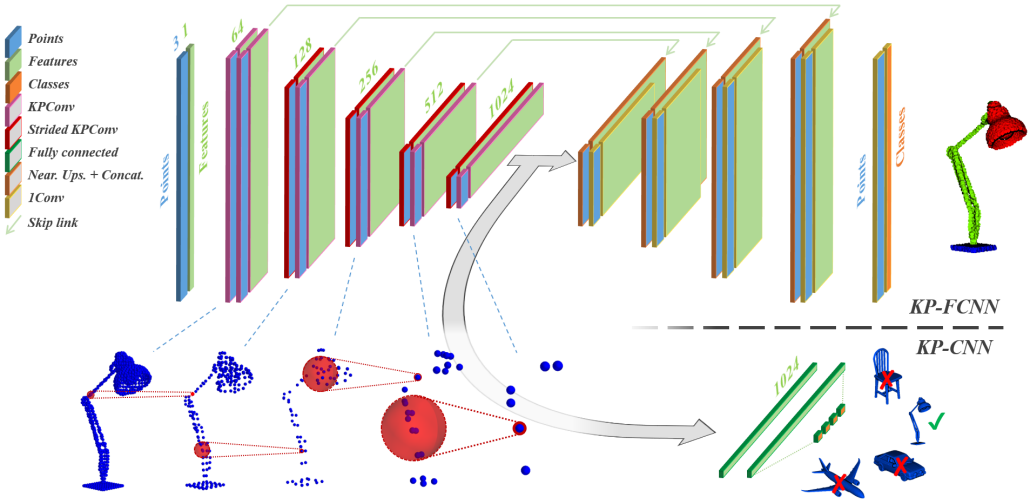


FIGURE 3.8 – Architecture de KP-FCNN et KP-CNN, basés sur des KPConvs.

3.4.4 Test de Kernel Point Fully Convolutional Neural Networks (KP-FCNNs)

Dans notre situation, nous avons la possibilité d'opter pour le réseau KP-FCNN, initialement conçu pour la segmentation, mais pouvant être ajusté pour la prédiction de bruit pour une utilisation dans DDPM. À cette fin, nous faisons usage de la bibliothèque easy_kpconv, implémentant les KPConvs et nous permettant de créer un code clair et réutilisable.

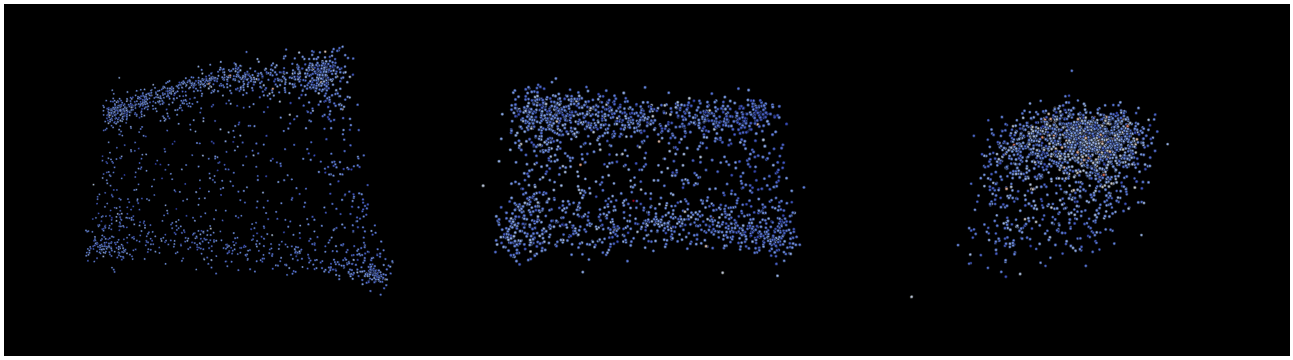


FIGURE 3.9 – Résultats d'un VDM KP-FCNN sur Rotor37_1200.

Comme illustré dans la Figure 3.9, lorsque cette architecture est associée à un modèle de diffusion, elle démontre la capacité de générer des nuages de points adoptant une structure en forme d'aube. Cependant, on remarque que les générations sont d'assez mauvaise qualité puisque les nuages de points semblent comporter une part de bruit résiduel importante. Certaines générations ne ressemblent même pas du tout à des aubes, mais plutôt à des nuages de points aléatoires. Autre point négatif, le decodeur de KP-FCNN étant composé de MLPs, il n'est pas scalable. En effet le réseau actuel comporte environ 4 millions de paramètres, dont une très large majorité se situent dans le décodeur.

3.4.5 Présentation des Point Voxel Convolutions (PVConvs)

Une seconde alternative aux opérations de convolutions et d'aggregation de PointNet++ sont les PVConvs [34]. Les PVConvs proposent d'utiliser des voxels afin de retomber sur des structures régulières, permettant ainsi d'effectuer efficacement des convolutions classiques en 3D. Par conséquent, une PVConv est composée de deux

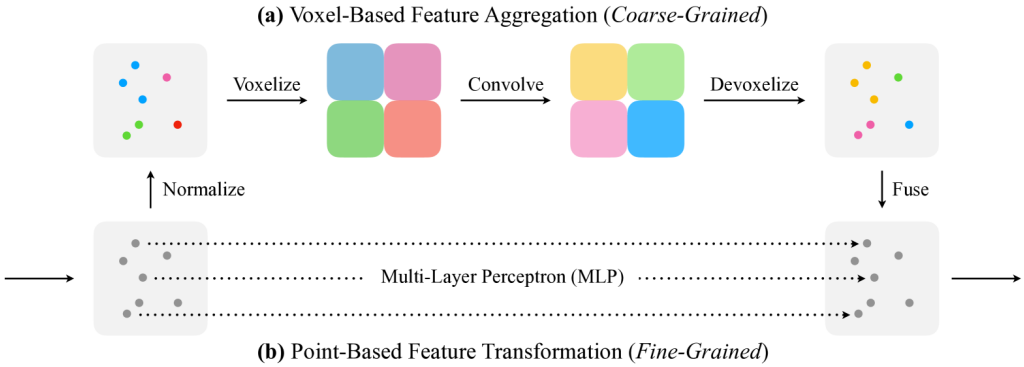


FIGURE 3.10 – Architecture d'une Point Voxel Convolution (PVConv).

branches distinctes, visibles sur la figure 3.10. La première branche, appelée "coarse-grained", transforme les points en voxels, applique des convolutions, puis reconvertit les voxels en points à l'aide d'une interpolation trilinéaire. La seconde branche, nommée "fine-grained", est constituée de shared MLPs. Ces deux branches sont ensuite combinées par sommation.

Par conséquent, PVCNN peut être considéré comme l'équivalent de PointNet, tandis que PVCNN++ correspond à PointNet++. Cependant, ces deux réseaux basés sur des PVConvs se distinguent par le maintien d'un nombre constant de points dans chaque couche du réseau. Les benchmarks démontrent que PVCNN++ est au moins aussi performant que PointNet++, tout en surpassant nettement ce dernier en termes d'efficacité globale. Cependant, il est important de noter que l'implémentation de PVCNN++ est assez complexe et nécessite l'utilisation de plusieurs modules CUDA (les opérations de voxelization et de dévoxelization étant impossible à écrire en PyTorch classique). Une version légèrement améliorée de PVConv, appelée Sparse Point Voxel Convolution (SPVConv), a également été développée en utilisant des opérations de convolution éparses. En effet, on observe empiriquement que les grilles de voxels sont fréquemment remplies en moyenne à hauteur de 60%. Cependant, cette amélioration nécessite l'utilisation de code CUDA très ésotérique.

3.4.6 Test de Point Voxel Diffusion (PVD)

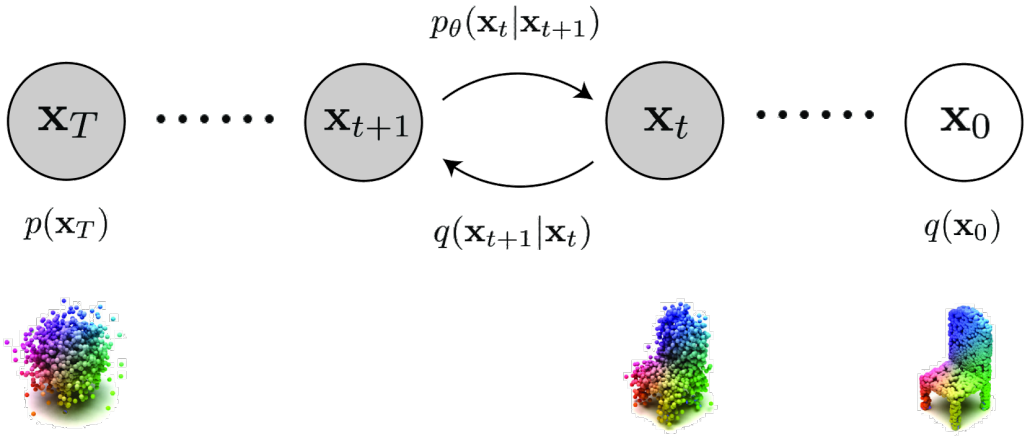


FIGURE 3.11 – Architecture de Point Voxel Diffusion (PVD).

Le premier papier ayant utilisé une architecture du type PVCNN++ pour la générations de nuages de point est PVD [65]. Ce réseau utilise DDPM et travaille directement sur le nuage de points. Si l'on récupère l'implémentation des auteurs et que l'on la modifie pour utiliser Rotor37_1200 on obtient de très bon résultat.

Cependant, une bonne partie de la codebase étant basé sur celle de PVCNN++ et de PointFlow, celle-ci est tout aussi difficile à modifier.

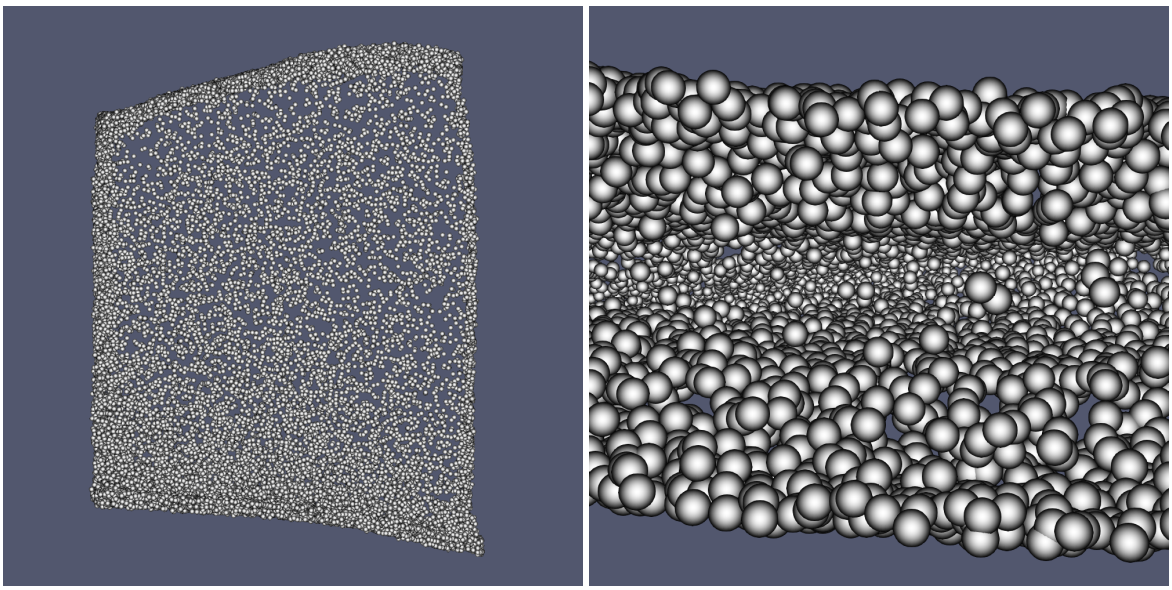


Figure 3.12 – Résultats de PVD sur Rotor37_1200.

Comme on l'observe sur la figure 3.12, les générations que produisent PVD sont de très bonne qualité. Contrairement à KP-FCNN, les générations ne présentent pas de bruits résiduels et la densité des points générés correspond à celle présente dans les données d'entrainement. On remarque tout de même quelques points anormales à l'intérieur de l'aube, ce qui pourrait impacter la reconstruction d'un maillage. De plus, PVD est un réseau assez lourd (27,6 millions de paramètres), et nécéssite un entrainement plutot long d'environ 10 heures pour produire de bon résultats.

3.4.7 Présentation de Latent Point Diffusion Model (LION)

LION [63] représente l'architecture la plus récente en matière de génération de nuages de points, et constitue une amélioration par rapport à PVD. LION exploite la diffusion latente via un VAE à deux étages pour transformer les nuages de points avant d'appliquer le processus de diffusion. L'architecture de LION est illustré sur la figure 3.13. Un premier encodeur transforme le nuage de points en un vecteur latent. Un deuxième encodeur transforme le nuage de points original, en utilisant le vecteur latent comme conditionnement, afin de générer un nuage de points latent. Ce nuage de points latent est ensuite décodé pour obtenir une reconstruction du nuage de point original. Le processus de diffusion a lieu à la fois sur le vecteur latent et sur le nuage de points latent.

LION est prometteur, cependant son entrainement est excessivement long, et peut prendre jusqu'à plusieurs semaines. Pour ces raisons, nous n'avons pas pu tester LION sur Rotor37_1200.

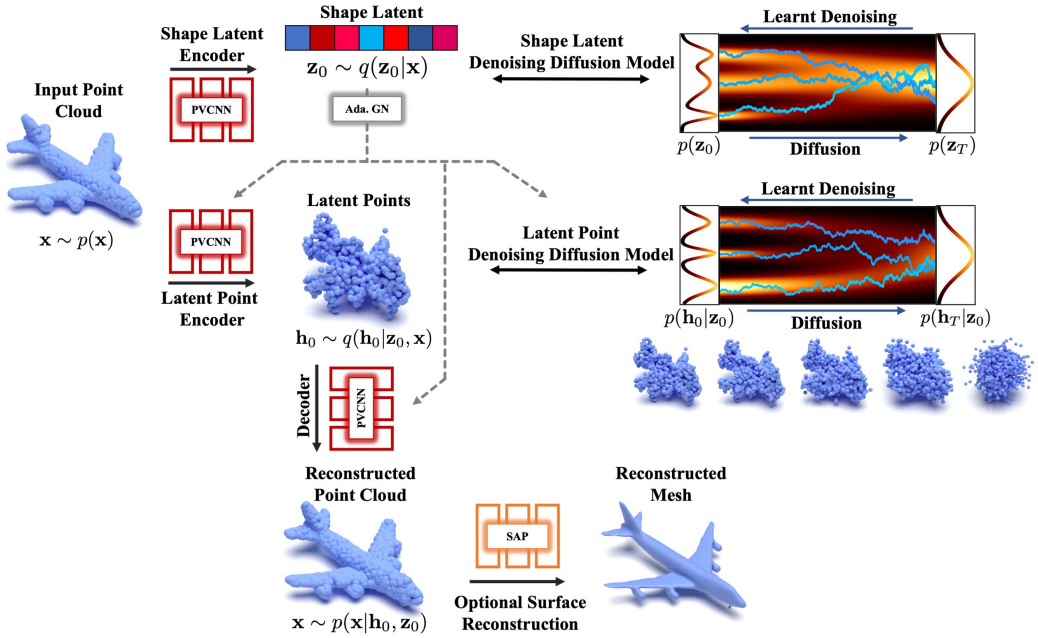


FIGURE 3.13 – Architecture de Latent Point Diffusion Model (LION) [63].

3.4.8 Synthèse des méthodes

Les résultats antérieurs démontrent la faisabilité de la résolution de notre problématique. Cependant, la plupart des approches précédentes se concentrent sur la génération non conditionnée ou utilisent des one-hot vectors pour le conditionnement sur des classes discrètes. Étant donné que notre objectif est de conditionner sur des métriques physiques scalaires, et étant donné que les implémentations précédemment présentées s'avèrent complexes à manipuler, nous avons pris la décision de développer notre propre réseau. De plus, il convient de noter que les modèles précédemment proposés sont souvent caractérisés par un nombre élevé de paramètres, ce qui s'avère excessif pour notre problématique, laquelle demeure comparativement plus simple.

Nous avons décidé d'adopter une approche reposant sur la réduction de dimension, via les LDM. Pour ce faire, il était crucial de choisir un modèle capable de transformer ou de réduire nos données d'entrée. Dans cette optique, nous avons opté pour une première étape de réduction de dimension par Principal Component Analysis (PCA). Il convient d'ailleurs de souligner que cette approche présente l'avantage d'être non paramétrique, éliminant ainsi la nécessité d'un processus d'optimisation.

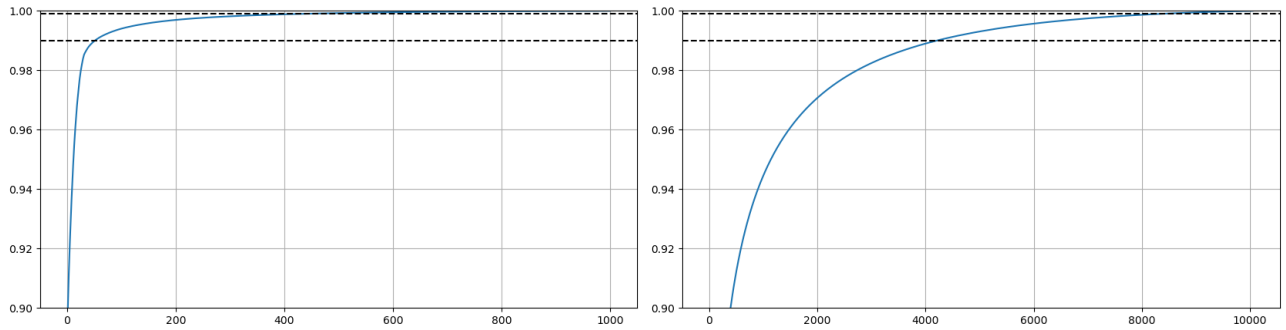


FIGURE 3.14 – Somme cumulative des modes de la PCA. Gauche: Rotor37_1200. Droite: Rotor37_11000.

À partir de la Figure 3.14, nous pouvons déduire qu'environ 30 modes PCA sont nécessaires pour représenter approximativement 99% de l'information présente dans le jeu de données Rotor37_1200. Cependant, il est à

noter que cette compression s'avère moins efficace pour le jeu de données Rotor37_11000, ce qui est attendu étant donné que les données présentent moins de similarités. Afin de résoudre les problèmes de compression associés au jeu de données Rotor37_11000, nous avons exploré la possibilité de remplacer la méthode PCA par une Proper Orthogonal Decomposition (POD). Malgré une meilleure capacité de compression obtenue par la POD, cette approche s'est révélée plus coûteuse en termes de calcul et moins adaptée aux autres techniques que nous prévoyons d'utiliser. En conséquence, pour le moment, nous avons décidé de maintenir l'utilisation de la PCA pour l'encodeur et de la "PCA inverse" pour le decodeur de notre LDM.

Dans le cas de Rotor37_1200, cela signifie que nous passons de 30000 points, chacun comportant à la fois leurs coordonnées spatiales et leurs normales (soit un total de 30000x6 scalaires), à un vecteur de taille 30. Ce vecteur étant petit, nous pouvons donc utiliser un simple MLP pour prédire le bruit dans le processus de diffusion. De plus, pour conditionner ce réseau en fonction de l'étape temporelle de la diffusion, nous pouvons simplement additionner les embeddings de pas de temps à chaque couche du réseau. Le réseau résultant contient à peine plus de 10000 paramètres, ce qui est considérablement inférieur aux autres modèles précédemment présentés. Par conséquent, ce réseau est très facile à manipuler et à entraîner, mais il présente l'inconvénient d'être spécialisé sur le jeu de données utilisé pour construire la PCA. Cependant, dans le contexte industriel, cet inconvénient est moins préoccupant car la nécessité de modèles génériques n'est pas primordiale.

Approche	**Qualité des générations**	**Paramètres**	**Temps d'entrainement**
GraphVAE	Correct, overfit, anomalies	4M	5m
PointFlow	DNF	1.3M	plusieurs jours
KP-FCNN	Correct, beacoup bruité	4M	1h
PVD	Bien, un peu bruité	27M	10h
LION	DNF	22.4M + 13.6M	plusieurs semaines
LDM PCA	Très Bien, peu générique	10k	20m

Table 3.1 – Tableau récapitulatif des différentes approches.

3.4.9 Conditionnement par Classifier-Free Guidance (CFG)

Pour conditionner notre processus de diffusion, nous avons opté pour l'utilisation de la CFG, avec une probabilité p_{uncond} de 10%. Pour intégrer les embeddings de conditionnement dans notre réseau, nous avons suivi une approche similaire à celle utilisée pour les embeddings de pas de temps. Les modifications apportées modifient très peu la surface de coût du réseau, et nous sommes désormais en mesure de générer des aubes tant non conditionnées que conditionnées, en fonction d'un vecteur scalaire de métriques physiques. Les résultats visuels des générations demeurent satisfaisants. Toutefois, il est difficile de déterminer visuellement si les aubes conditionnées générées respectent effectivement les critères de conditionnement qui leur sont assignés.

La figure 3.15 montre les résultats d'un LDM PCA sur Rotor37_1200:

- Blanc: génération non conditionnée
- Rouge: génération conditionnée, isentropic_effiency=1, $\gamma = 7$
- Jaune: génération conditionnée, isentropic_effiency=1, $\gamma = 14$
- Bleu: génération conditionnée, isentropic_effiency=1, $\gamma = 25$

On observe qu'à mesure que le facteur de guidage γ augmente, certaines parties de l'aube subissent une déformation plus prononcée. À des fins d'illustration, il est important de noter que les valeurs de γ utilisées ici sont considérablement élevées. En pratique, toutefois, γ ne dépasse pas généralement la valeur de 5, et demeure en moyenne autour de 2.

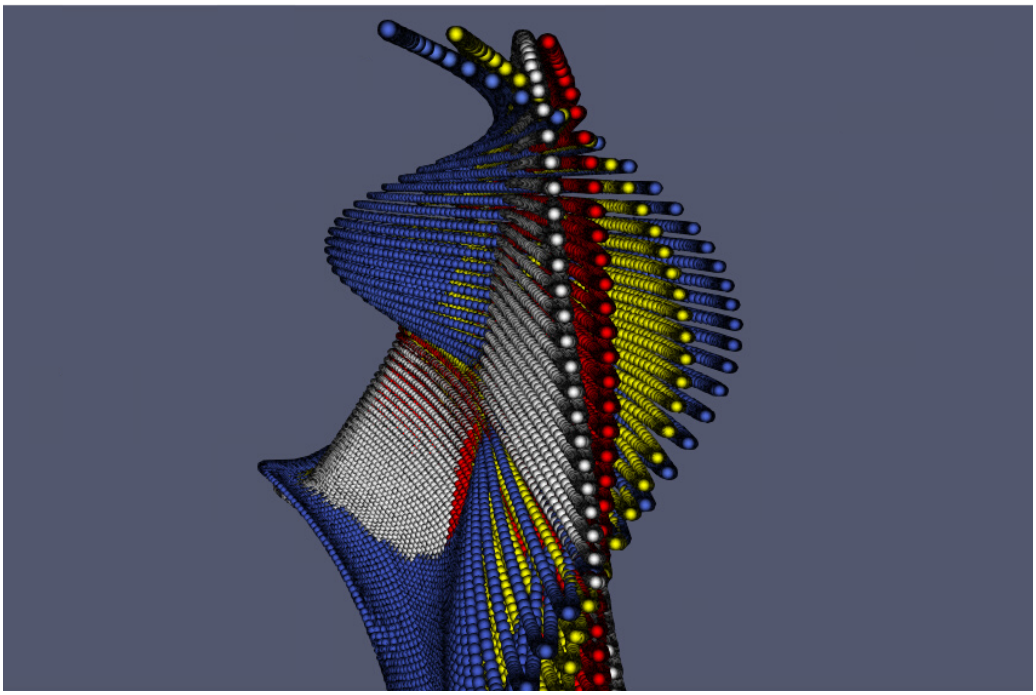


FIGURE 3.15 – Résultats d'un LDM PCA sur Rotor37_1200, pour plusieurs valeurs de γ.

3.4.10 Vérification par Gaussian Process (GP)

Dans cette optique, il devient impératif de mettre en place un moyen de vérification des propriétés physiques des aubes générées et de les comparer aux critères de conditionnement. Bien que l'approche la plus simple consisterait à soumettre nos générations à un simulateur CFD, cette méthode s'avère inenvisageable en raison de sa lenteur.

Une solution viable consiste à entraîner un modèle de régression capable d'établir une relation entre nos nuages de points (ou leurs espaces latents) et leurs critères de performances. Cette tâche peut être réalisée en utilisant des réseaux de neurones, mais également en optant pour une approche moins paramétrique telle que les GPs.

Les GPs sont des modèles probabilistes non paramétriques qui trouvent leur application dans des tâches de régression ou de classification. Ils sont particulièrement adaptés aux tâches de régression, car ils sont capables de fournir des estimations de l'incertitude associée à leurs prédictions. Cette mesure d'incertitude se révèle particulièrement précieuse pour la vérification de nos aubes générées, car elle permet d'évaluer la validité de nos générations en fournissant une estimation de la fiabilité de ces dernières.

Ainsi si l'on entraine un GP sur des nuages de points avec leur critères de performance associés, on peut ensuite utiliser ce GP pour vérifier si nos nuages de points générés sont corrects. En effet, si l'on génère un nuage de points, et que l'on obtient un critère de performance très différent de celui attendu, alors on peut en déduire que notre génération est incorrecte.

La figure 3.17 illustre la validation du GP en utilisant des données de test. En traçant la prédiction du GP par rapport aux valeurs réelles, on observe une relation linéaire avec une pente de 1. Cette observation est confirmée numériquement par la proximité de Q^2 à 1. Ces résultats indiquent que le GP est capable de prédire les indicateurs de performance des aubes, en se basant sur leurs modes PCA.

De manière complémentaire, il est intéressant de noter que les distributions de in_massflow, out_massflow et

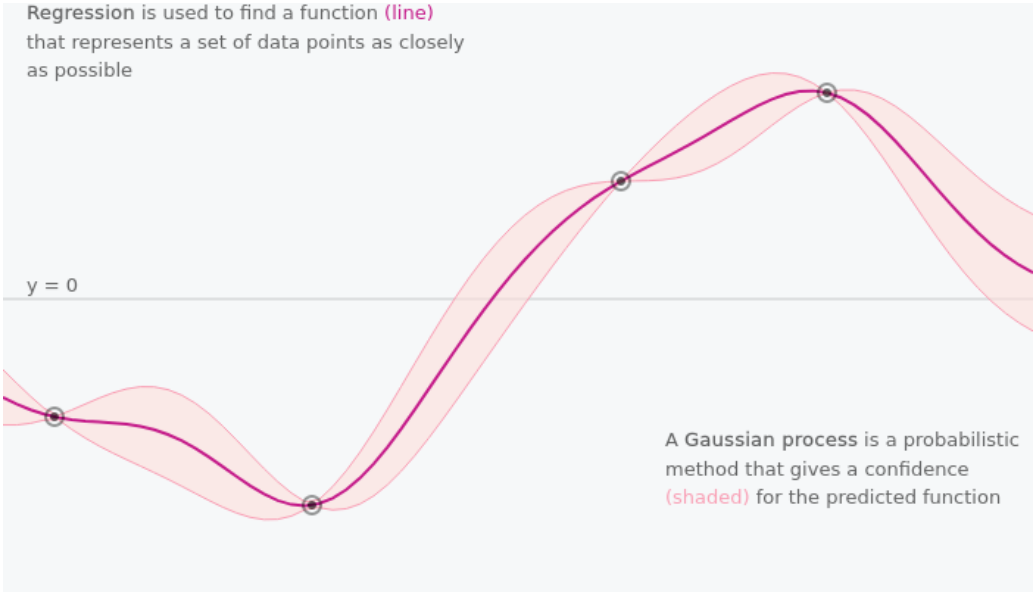


FIGURE 3.16 – Régression par GP.
Source: distill.pub [19].

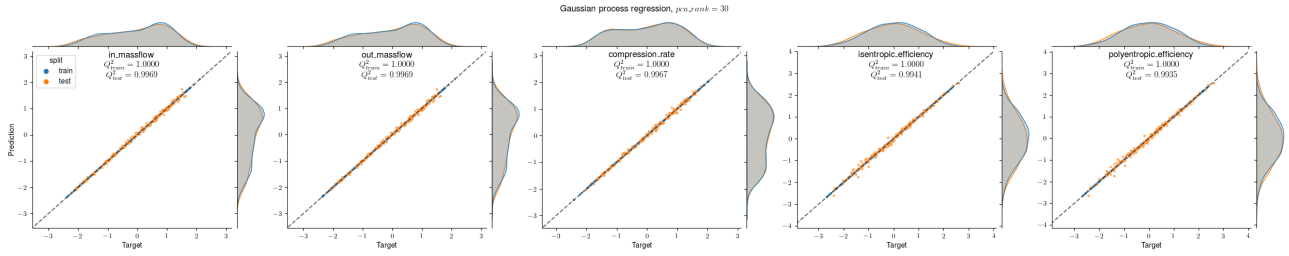


FIGURE 3.17 – Entrainement d'un GP sur 30 modes PCA de Rotor37_1200.

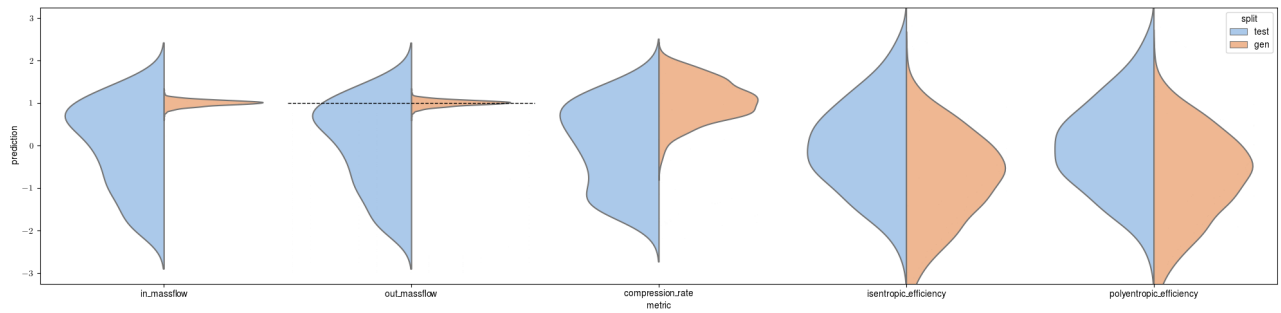


FIGURE 3.18 – Vérification du conditionnement (out_massflow=1) par GP.

compression_rate présentent une grande similitude. Cette similarité est attendu, étant donné la forte corrélation physique entre ces trois critères. Un schéma similaire se manifeste entre les distributions de isentropic_efficiency et polyentropic_efficiency.

Comme illustré dans les figures 3.18 et 3.19, il est nettement observable qu'il y a un changement dans la densité de probabilité des critères de performance lorsque nous conditionnons nos générations. Cette observation suggère que le conditionnement du modèle a effectivement un impact sur les critères de performance, selon le GP. Cependant, pour confirmer cette hypothèse, il faudrait de procéder à une simulation numérique CFD. En

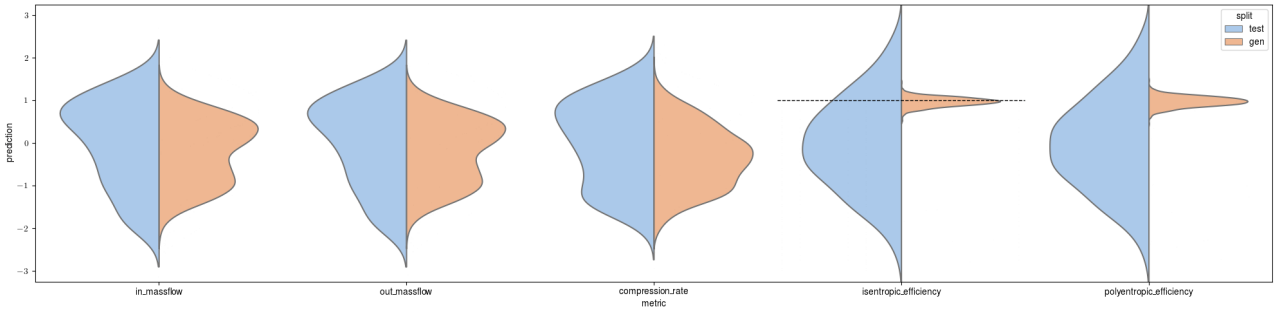


FIGURE 3.19 – Vérification du conditionnement (isentropy_efficiency=1) par GP.

outre, on peut également constater sa capacité à générer des données qui se situent en dehors de la distribution d'entraînement, à condition que ces données demeurent relativement proches de ladite distribution. Par exemple, il est totalement inenvisageable de demander la génération d'aubes ayant des valeurs différentes pour les attributs in_massflow et out_massflow. Cette impossibilité résulte du fait que de telles combinaisons ne se trouvent pas dans le jeu de données d'entraînement, et qu'elles sont également physiquement incohérentes dans un système clos tel qu'un moteur d'avion.

Chapitre 4

Conclusion

En conclusion, l'étude de ce stage a démontré avec succès la faisabilité de la génération conditionnée d'aubes de turbines en utilisant une approche basée sur des modèles de diffusion. Les résultats obtenus ont dépassé nos attentes initiales en matière de précision et de capacité à générer des aubes conditionnées tout en préservant les propriétés physiques essentielles.

Sur le plan personnel, cette recherche m'a offert une occasion précieuse d'approfondir mes compétences en matière de modélisation générative, en explorant l'application des DDPMs à un problème concret de l'ingénierie. Du point de vue des connaissances techniques, j'ai pu consolider mes connaissances en matière de réseaux de neurones, d'utilisation de bibliotèques de deep learning, ainsi que dans le domaine des techniques classiques de machine learning telles que la PCA ou les GPs.

Enfin, ce travail de recherche a également mis en lumière l'importance de la collaboration entre les domaines techniques et académiques. Mes échanges et discussions avec les ingénieurs, les experts du domaine industriel et certains auteurs de papiers ont joué un rôle essentiel dans l'orientation de ma recherche et dans la validation des résultats.

En perspective d'amélioration, il serait pertinent d'explorer des méthodes visant à réduire la taille des maillages utilisés dans les simulations numériques, tout en maintenant leur précision. Cette approche pourrait éventuellement conduire à l'obtention de modèles plus compacts et plus rapides à entraîner, tout en préservant une représentation précise des propriétés physiques des aubes. Par ailleurs, il serait également intéressant d'explorer la possibilité d'appliquer ces techniques de génération directement aux CAOs, sans passer par les étapes intermédiaires des maillages ou des nuages de points. Cette démarche pourrait simplifier le processus de conception en générant directement des modèles CAO conditionnés selon les spécifications requises. Enfin, il serait aussi pertinent d'essayer des méthodes par Reinforcement Learning (RL) sur ce problème, car ces méthodes ont souvent de très bonne capacités de généralisation.

Bibliographie

[1] James Ahrens, Berk Geveci, and Charles Law. ParaView: An end-user tool for large data visualization. In *Visualization Handbook*. Elesvier, 2005. URL http://www.paraview.org/. ISBN 978-0123875822. 21

[2] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep Variational Information Bottleneck, October 2019. URL http://arxiv.org/abs/1612.00410. arXiv:1612.00410 [cs, math]. 11

[3] Hector Andrade-Loarca, Julius Hege, Aras Bacho, and Gitta Kutyniok. PoissonNet: Resolution-Agnostic 3D Shape Reconstruction using Fourier Neural Operators, August 2023. URL http://arxiv.org/abs/2308.01766. arXiv:2308.01766 [cs, math]. 24

[4] Martin Arjovsky and Léon Bottou. Towards Principled Methods for Training Generative Adversarial Networks, January 2017. URL http://arxiv.org/abs/1701.04862. arXiv:1701.04862 [cs, stat]. 10

[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN, December 2017. URL http://arxiv.org/abs/1701.07875. arXiv:1701.07875 [cs, stat]. 10

[6] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016. 12

[7] Shaked Brody, Uri Alon, and Eran Yahav. How Attentive are Graph Attention Networks?, January 2022. URL http://arxiv.org/abs/2105.14491. arXiv:2105.14491 [cs]. 6

[8] Christopher P. Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-VAE, April 2018. URL http://arxiv.org/abs/1804.03599. arXiv:1804.03599 [cs, stat]. 11

[9] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 12

[10] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A Universe of Annotated 3D Objects, December 2022. URL http://arxiv.org/abs/2212.08051. arXiv:2212.08051 [cs]. 12

[11] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, Eli VanderBilt, Aniruddha Kembhavi, Carl Vondrick, Georgia Gkioxari, Kiana Ehsani, Ludwig Schmidt, and Ali Farhadi. Objaverse-XL: A Universe of 10M+ 3D Objects, July 2023. URL http://arxiv.org/abs/2307.05663. arXiv:2307.05663 [cs]. 12

[12] Prafulla Dhariwal and Alex Nichol. Diffusion Models Beat GANs on Image Synthesis, June 2021. URL http://arxiv.org/abs/2105.05233. arXiv:2105.05233 [cs, stat]. 12, 15

[13] Carl Doersch. Tutorial on Variational Autoencoders, January 2021. URL http://arxiv.org/abs/1606.05908. arXiv:1606.05908 [cs, stat]. 10

[14] Faezeh Faez, Yassaman Ommi, Mahdieh Soleymani Baghshah, and Hamid R. Rabiee. Deep Graph Generators: A Survey, December 2020. URL http://arxiv.org/abs/2012.15544. arXiv:2012.15544 [cs]. 9

[15] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, and Yong Li. A Survey of Graph Neural Networks for Recommender Systems: Challenges, Methods, and Directions, January 2023. URL http://arxiv.org/abs/2109.12843. arXiv:2109.12843 [cs]. 6

[16] Hongyang Gao and Shuiwang Ji. Graph U-Nets, May 2019. URL http://arxiv.org/abs/1905.05178. arXiv:1905.05178 [cs, stat]. 6, 24

[17] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks, June 2014. URL http://arxiv.org/abs/1406.2661. arXiv:1406.2661 [cs, stat]. 9

[18] Xiaojie Guo and Liang Zhao. A Systematic Survey on Deep Generative Models for Graph Generation, October 2022. URL http://arxiv.org/abs/2007.06686. arXiv:2007.06686 [cs, stat]. 9

[19] Jochen Görtler, Rebecca Kehlbeck, and Oliver Deussen. A visual exploration of gaussian processes. *Distill*, 2019. doi: 10.23915/distill.00017. https://distill.pub/2019/visual-exploration-gaussian-processes. 33

[20] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. July 2022. URL https://openreview.net/forum?id=Sy2fzU9gl. 11

[21] Jonathan Ho and Tim Salimans. Classifier-Free Diffusion Guidance, July 2022. URL http://arxiv.org/abs/2207.12598. arXiv:2207.12598 [cs]. 16

[22] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models, December 2020. URL http://arxiv.org/abs/2006.11239. arXiv:2006.11239 [cs, stat]. 13

[23] Jinwoo Kim, Jaehoon Yoo, Juho Lee, and Seunghoon Hong. SetVAE: Learning Hierarchical Composition for Generative Modeling of Set-Structured Data, March 2021. URL http://arxiv.org/abs/2103.15619. arXiv:2103.15619 [cs]. 11

[24] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes, December 2022. URL http://arxiv.org/abs/1312.6114. arXiv:1312.6114 [cs, stat]. 10

[25] Thomas Kipf, Petar Veličković, and Yujia Li. Graph Neural Networks for Modeling Small Molecules. March 2020. 6

[26] Thomas N. Kipf and Max Welling. Variational Graph Auto-Encoders, November 2016. URL http://arxiv.org/abs/1611.07308. arXiv:1611.07308 [cs, stat]. 10

[27] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks, February 2017. URL http://arxiv.org/abs/1609.02907. arXiv:1609.02907 [cs, stat]. 6

[28] Ivan Kobyzev, Simon J. D. Prince, and Marcus A. Brubaker. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11): 3964–3979, November 2021. ISSN 0162-8828, 2160-9292, 1939-3539. doi: 10.1109/TPAMI.2020.2992934. URL http://arxiv.org/abs/1908.09257. arXiv:1908.09257 [cs, stat]. 11

[29] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, May 2017. URL http://arxiv.org/abs/1609.04802. arXiv:1609.04802 [cs, stat]. 10

[30] Quentin Lhoest and The HuggingFace team. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. URL https://github.com/huggingface/datasets. 22

[31] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks, September 2017. URL http://arxiv.org/abs/1511.05493. arXiv:1511.05493 [cs, stat]. 6

[32] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard S. Zemel. Efficient Graph Generation with Graph Recurrent Attention Networks, July 2020. URL http://arxiv.org/abs/1910.00760. arXiv:1910.00760 [cs, stat]. 18

[33] Yiyi Liao, Jun Xie, and Andreas Geiger. KITTI-360: A Novel Dataset and Benchmarks for Urban Scene Understanding in 2D and 3D, June 2022. URL http://arxiv.org/abs/2109.13410. arXiv:2109.13410 [cs]. 12

[34] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-Voxel CNN for Efficient 3D Deep Learning, December 2019. URL http://arxiv.org/abs/1907.03739. arXiv:1907.03739 [cs]. 27

[35] Calvin Luo. Understanding Diffusion Models: A Unified Perspective, August 2022. URL http://arxiv.org/abs/2208.11970. arXiv:2208.11970 [cs]. v, 13, 14

[36] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z. Sheng, Hui Xiong, and Leman Akoglu. A Comprehensive Survey on Graph Anomaly Detection with Deep Learning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021. ISSN 1041-4347, 1558-2191, 2326-3865. doi: 10.1109/TKDE.2021.3118815. URL http://arxiv.org/abs/2106.07178. arXiv:2106.07178 [cs]. 6

[37] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, August 2020. URL http://arxiv.org/abs/2003.08934. arXiv:2003.08934 [cs]. 18

[38] S. Mouriaux, F. Bassi, A. Colombo, and A. Ghidoni. NASA Rotor 37. In Charles Hirsch, Koen Hillewaert, Ralf Hartmann, Vincent Couaillier, Jean-Francois Boussuge, Frederic Chalot, Sergey Bosniakov, and Werner Haase, editors, *TILDA: Towards Industrial LES/DNS in Aeronautics: Paving the Way for Future Accurate CFD - Results of the H2020 Research Project TILDA, Funded by the European Union, 2015 -2018*, Notes on Numerical Fluid Mechanics and Multidisciplinary Design, pages 533–544. Springer International Publishing, Cham, 2021. ISBN 978-3-030-62048-6. doi: 10.1007/978-3-030-62048-6_20. URL https://doi.org/10.1007/978-3-030-62048-6_20. 21

[39] Gimin Nam, Mariem Khlifi, Andrew Rodriguez, Alberto Tono, Linqi Zhou, and Paul Guerrero. 3D-LDM: Neural Implicit 3D Shape Generation with Latent Diffusion Models, December 2022. URL http://arxiv.org/abs/2212.00842. arXiv:2212.00842 [cs]. 19

[40] Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter W. Battaglia. PolyGen: An Autoregressive Generative Model of 3D Meshes, February 2020. URL http://arxiv.org/abs/2002.10880. arXiv:2002.10880 [cs, stat]. 18

[41] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models, March 2022. URL http://arxiv.org/abs/2112.10741. arXiv:2112.10741 [cs]. 16

[42] Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-E: A System for Generating 3D Point Clouds from Complex Prompts, December 2022. URL http://arxiv.org/abs/2212.08751. arXiv:2212.08751 [cs]. 19

[43] Augustus Odena. Semi-Supervised Learning with Generative Adversarial Networks, October 2016. URL http://arxiv.org/abs/1606.01583. arXiv:1606.01583 [cs, stat]. 10

[44] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. URL https://distill.pub/2017/feature-visualization. 11

[45] Xingang Pan, Ayush Tewari, Thomas Leimkühler, Lingjie Liu, Abhimitra Meka, and Christian Theobalt. Drag Your GAN: Interactive Point-based Manipulation on the Generative Image Manifold, May 2023. URL http://arxiv.org/abs/2305.10973. arXiv:2305.10973 [cs]. 10

[46] Songyou Peng, Chiyu "Max" Jiang, Yiyi Liao, Michael Niemeyer, Marc Pollefeys, and Andreas Geiger. Shape As Points: A Differentiable Poisson Solver, November 2021. URL http://arxiv.org/abs/2106.03452. arXiv:2106.03452 [cs]. 24

[47] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation, April 2017. URL http://arxiv.org/abs/1612.00593. arXiv:1612.00593 [cs]. 25

[48] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space, June 2017. URL http://arxiv.org/abs/1706.02413. arXiv:1706.02413 [cs]. 25

[49] Neal Richardson, Ian Cook, Nic Crane, Dewey Dunnington, Romain François, Jonathan Keane, Dragoș Moldovan-Grünfeld, Jeroen Ooms, and Apache Arrow. *arrow: Integration to 'Apache' 'Arrow'*, 2023. URL https://github.com/apache/arrow/. 22

[50] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models, April 2022. URL http://arxiv.org/abs/2112.10752. arXiv:2112.10752 [cs]. v, 15

[51] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs, June 2016. URL http://arxiv.org/abs/1606.03498. arXiv:1606.03498 [cs]. 10

[52] Sohil Atul Shah and Vladlen Koltun. Auto-decoding Graphs, June 2020. URL http://arxiv.org/abs/2006.02879. arXiv:2006.02879 [cs, stat]. 11

[53] Martin Simonovsky and Nikos Komodakis. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders, February 2018. URL http://arxiv.org/abs/1802.03480. arXiv:1802.03480 [cs]. 6, 11, 23

[54] Yang Song and Stefano Ermon. Generative Modeling by Estimating Gradients of the Data Distribution, October 2020. URL http://arxiv.org/abs/1907.05600. arXiv:1907.05600 [cs, stat]. 14, 15

[55] Jianlin Su and Guang Wu. f-VAEs: Improve VAEs with Conditional Flows, September 2018. URL http://arxiv.org/abs/1809.05861. arXiv:1809.05861 [cs, stat]. 11

[56] Raphael Sulzer, Loic Landrieu, Alexandre Boulch, Renaud Marlet, and Bruno Vallet. Deep Surface Reconstruction from Point Clouds with Visibility Information, February 2022. URL http://arxiv.org/abs/2202.01810. arXiv:2202.01810 [cs]. 24

[57] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes, January 2021. URL http://arxiv.org/abs/2101.10994. arXiv:2101.10994 [cs]. 19

[58] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds, August 2019. URL http://arxiv.org/abs/1904.08889. arXiv:1904.08889 [cs]. 26

[59] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks, February 2018. URL http://arxiv.org/abs/1710.10903. arXiv:1710.10903 [cs, stat]. 6

[60] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes, April 2015. URL http://arxiv.org/abs/1406.5670. arXiv:1406.5670 [cs]. 12

[61] Yaniv Yacoby, Weiwei Pan, and Finale Doshi-Velez. Failure Modes of Variational Autoencoders and Their Effects on Downstream Tasks. March 2021. URL https://openreview.net/forum?id=5Spjp0zDYt. 11

[62] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows, September 2019. URL http://arxiv.org/abs/1906.12320. arXiv:1906.12320 [cs]. 12

[63] Xiaohui Zeng, Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, and Karsten Kreis. LION: Latent Point Diffusion Models for 3D Shape Generation, October 2022. URL http://arxiv.org/abs/2210.06978. arXiv:2210.06978 [cs, stat]. vi, 29, 30

[64] Shengjia Zhao, Hongyu Ren, Arianna Yuan, Jiaming Song, Noah Goodman, and Stefano Ermon. Bias and Generalization in Deep Generative Models: An Empirical Study, November 2018. URL http://arxiv.org/abs/1811.03259. arXiv:1811.03259 [cs, stat]. 10

[65] Linqi Zhou, Yilun Du, and Jiajun Wu. 3D Shape Generation and Completion through Point-Voxel Diffusion, August 2021. URL http://arxiv.org/abs/2104.03670. arXiv:2104.03670 [cs]. 28

[66] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, August 2020. URL http://arxiv.org/abs/1703.10593. arXiv:1703.10593 [cs]. 10

[67] Yanqiao Zhu, Yuanqi Du, Yinkai Wang, Yichen Xu, Jieyu Zhang, Qiang Liu, and Shu Wu. A Survey on Deep Graph Generation: Methods and Applications, December 2022. URL http://arxiv.org/abs/2203.06714. arXiv:2203.06714 [cs, q-bio]. 9