Rapport de Projet : Génération d'Images par Modèles de Diffusion

Tanguy Ducroq, Guillaume Rodriguez

11 Janvier 2026

1 Introduction

Ce projet a pour objectif d'implémenter et d'entraîner des modèles de diffusion (DDPM) "from scratch" pour la génération d'images. Notre démarche a été incrémentale, partant d'architectures simples pour valider les concepts mathématiques sur des données basiques, jusqu'à des modèles conditionnels complexes capables de générer des avatars de type "CryptoPunks" avec des attributs spécifiques (genre, accessoires, etc.).

Nous avons également développé une interface de supervision et de génération interactive via Streamlit pour faciliter l'exploration des résultats.

2 Méthodologie Progressive

Notre approche s'est découpée en quatre phases distinctes pour isoler les difficultés :

2.1 Phase 1 : Diffusion Simple sur MNIST

Nous avons commencé par implémenter une boucle d'entraînement de diffusion classique (DDPM) couplée à une architecture U-Net simplifiée. L'objectif était de valider la pipeline :

- Ajout de bruit progressif (schedule linéaire).
- Entraînement du modèle à prédire le bruit ajouté.
- Sampling inverse pour générer des chiffres.

Cette étape a confirmé que notre U-Net et notre gestion du bruit fonctionnaient correctement sur des images 16×16 en noir et blanc.

2.2 Phase 2 : Passage aux CryptoPunks (Non-Conditionnel)

Une fois le concept validé, nous avons adapté le modèle aux images RGB des CryptoPunks. La complexité a augmenté (3 canaux couleurs, plus de détails), nécessitant un ajustement de la capacité du réseau (nombre de filtres, profondeur). À ce stade, le modèle générait des visages aléatoires sans aucun contrôle sur le résultat.

2.3 Phase 3 : Conditionnement par Classes (MNIST)

Nous avons ensuite introduit le conditionnement pour contrôler la génération (e.g., demander un "3" ou un "7"). C'est ici que nous avons expérimenté l'injection d'embeddings de classes dans le U-Net, concaténés avec les embeddings temporels.

2.4 Phase 4 : Conditionnement Multi-Attributs (CryptoPunks)

Enfin, nous avons généralisé l'approche pour les CryptoPunks en utilisant les métadonnées (Json). Contrairement à MNIST (une seule classe par image), les CryptoPunks possèdent de multiples attributs (Type : Male/Female, Accessoires : Glasses, Hat, etc.). Nous avons créé un vecteur d'embedding combinant ces caractéristiques pour guider la diffusion.

3 Difficultés Techniques et Solutions (Le "Vrai" CFG)

La principale difficulté rencontrée concernait le conditionnement du modèle, particulièrement pour les CryptoPunks.

3.1 L'échec de l'embedding simple

Initialement, nous avons naïvement injecté l'embedding des classes (ou attributs) directement dans le réseau, en l'additionnant ou le concaténant simplement à l'embedding de temps. **Problème :** Le réseau avait tendance à ignorer cette information subtile. Bien que la perte (loss) diminuait, le modèle peinait à respecter strictement la consigne (par exemple, générer un homme avec des lunettes produisait souvent un résultat flou ou sans lunettes), l'information de guidage étant "noyée" dans le bruit.

3.2 La solution : Classifier-Free Guidance (CFG)

Pour pallier ce manque d'adhérence aux consignes, nous avons implémenté le *Classifier-Free Guidance*. Au lieu de simplement passer l'embedding de condition c, la méthode consiste à entraîner le modèle conjointement en conditionnel et en inconditionnel (en remplaçant aléatoirement le label par un vecteur vide lors de l'entraînement).

Lors de la génération (sampling), nous calculons deux prédictions de bruit :

— ϵ_{cond} : La prédiction sachant les attributs (avec guidage).

— ϵ_{uncond} : La prédiction "libre" (sans guidage).

Le bruit final est une extrapolation linéaire :

$$\epsilon_{final} = \epsilon_{uncond} + w \cdot (\epsilon_{cond} - \epsilon_{uncond})$$

Avec un poids $w > 1$, nous forçons le modèle à amplifier les caractéristiques spécifiques demandées. Cela a radicalement amélioré la précision des attributs générés.

4 Interface et Supervision

Pour superviser les entraînements souvent longs, nous avons développé une application **Streamlit** (`streamlit_dashboard.py`).

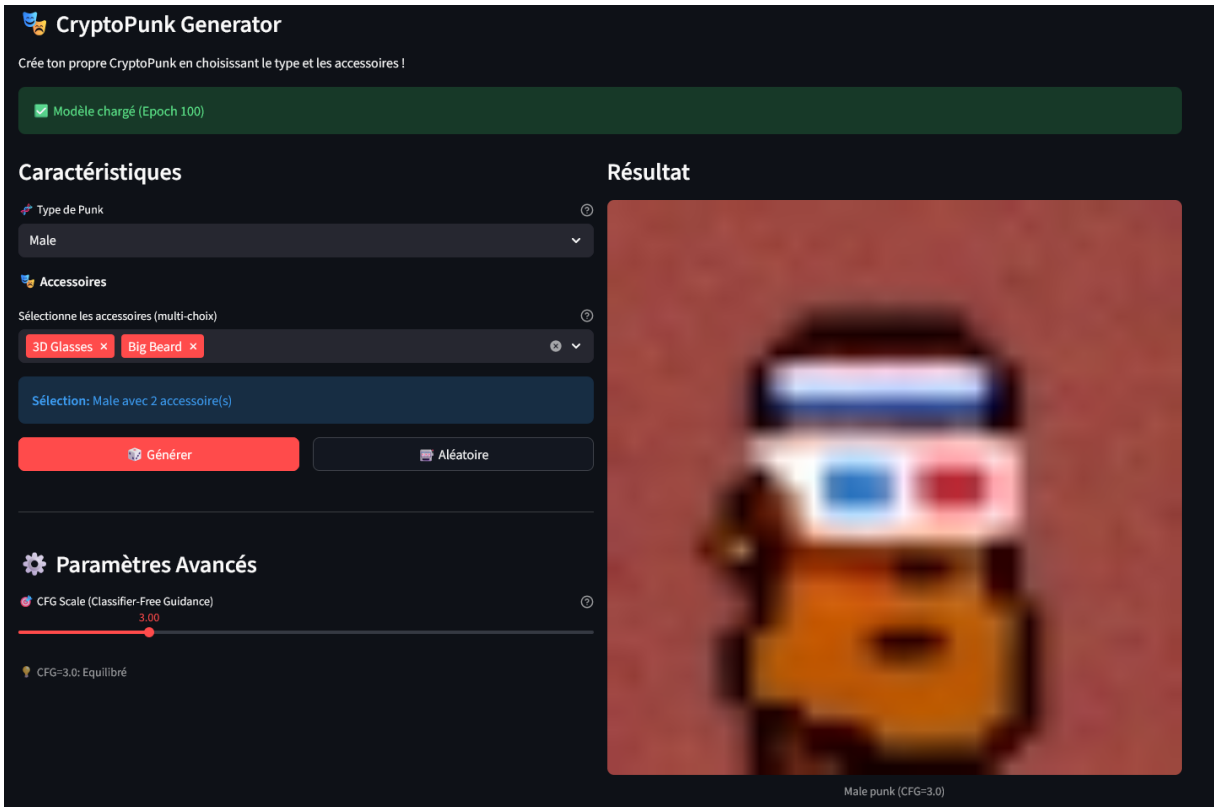

FIGURE 1 – Interface de supervision permettant la génération interactive guidée

Cette interface permet de :

1. Suivre en temps réel les courbes de Loss (données Tensorboard).
2. Visualiser les samples générés à chaque époque pour vérifier la convergence.
3. Tester le modèle final en mode interactif : l'utilisateur coche des cases (Homme, Lunettes noires, Chapeau...) et le modèle génère l'image correspondante à la volée en utilisant le CFG.

5 Résultats

Ci-dessous un aperçu des résultats obtenus sur les configurations finales.

5.1 Évolution de l'apprentissage

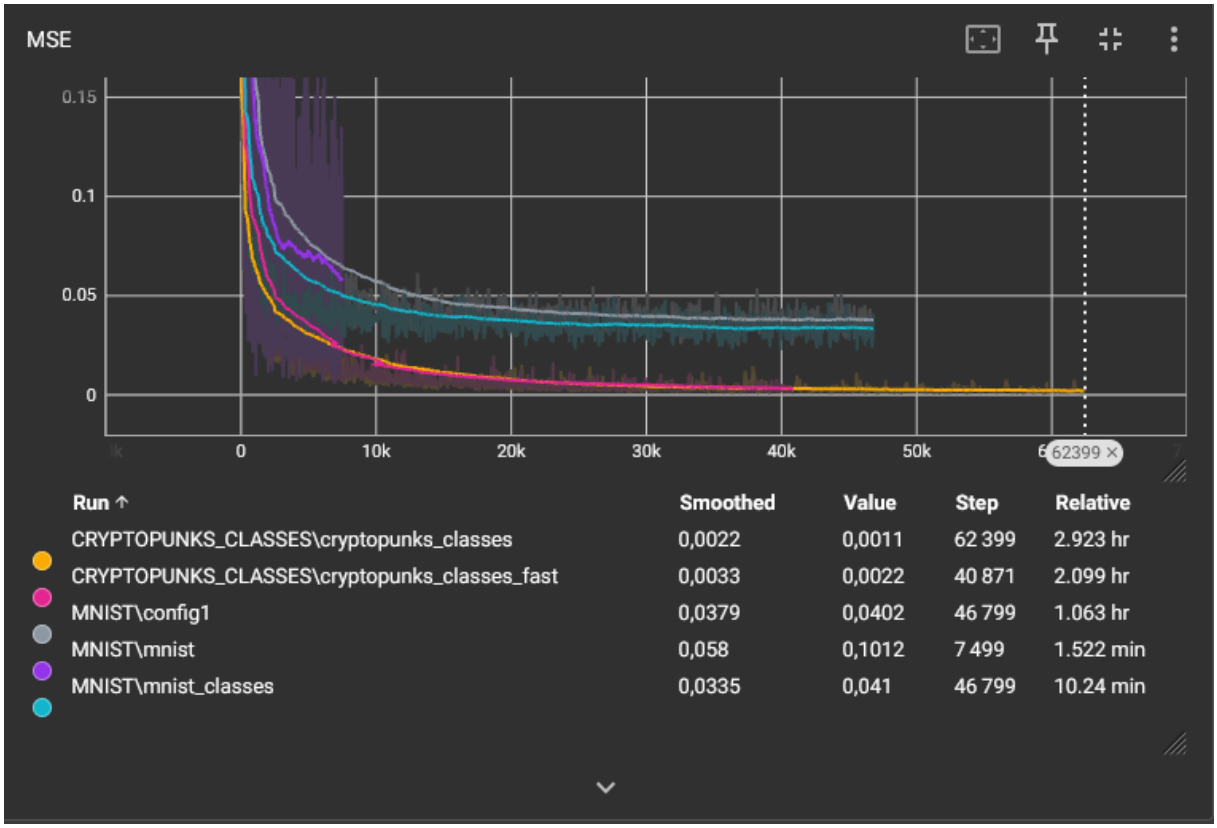

FIGURE 2 – Comparaison des courbes de perte (Loss) pour les différents entraînements

5.2 Générations Conditionnelles

5.2.1 MNIST

Le modèle parvient à générer des chiffres spécifiques avec une grande clarté.

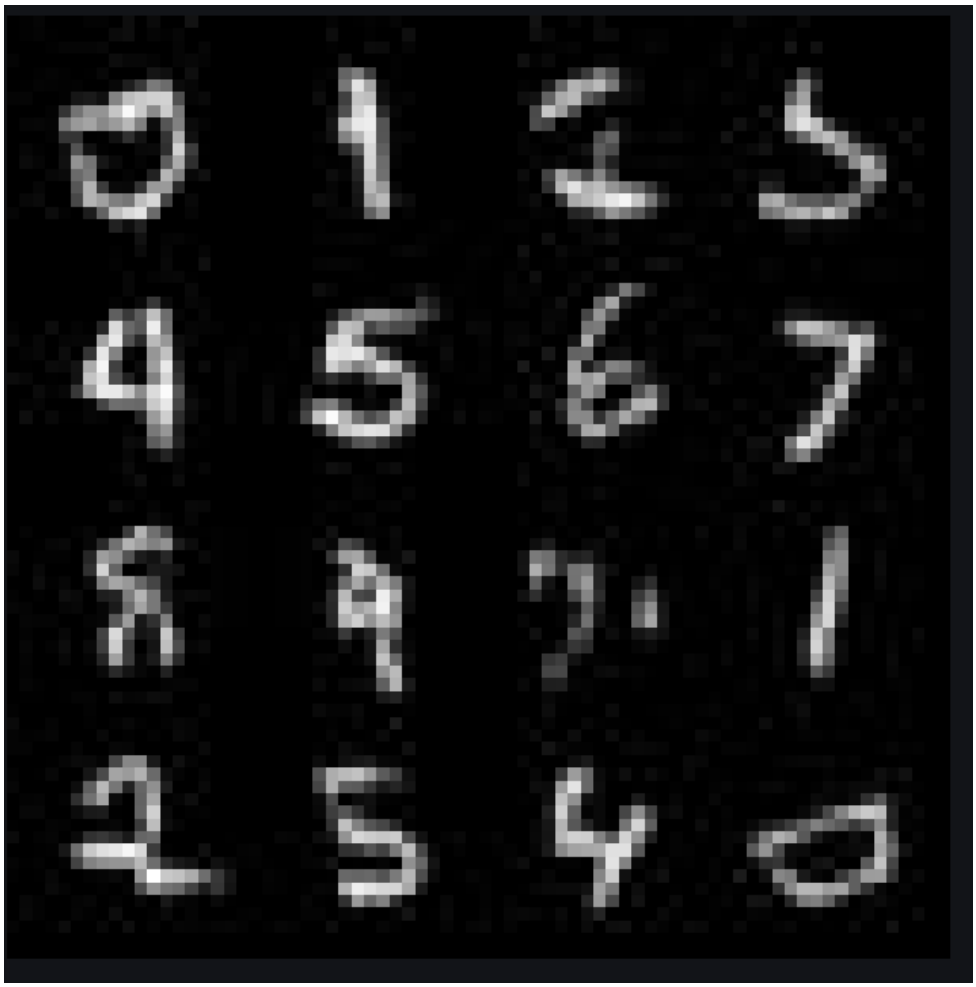

FIGURE 3 – Génération de chiffres MNIST conditionnés par classes

5.2.2 CryptoPunks (Multi-Attributs)

Grâce à l'approche accélérée et au CFG, nous obtenons des résultats cohérents respectant les contraintes d'attributs.

FIGURE 4 – CryptoPunks générés par le modèle final