



# *Adversarial Diffusion Distillation*

## Pokémon Finetuning



cy

Auteur : *[Bilal El Biyadi, Yvan Louamba]*  
Professeur : *Arthur SARMINI DET SATOUF*  
Établissement : *CY TECH*

13 avril 2025

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>L'approche ADD</b>	<b>3</b>
2.1	Formulation mathématique . . . . .	3
2.2	Stratégie d'entraînement . . . . .	4
2.3	Usage en inférence . . . . .	4
2.4	Résultats et analyses . . . . .	4
<b>3</b>	<b>Dataset</b>	<b>4</b>
3.1	Images . . . . .	4
3.2	Dictionnaires . . . . .	5
<b>4</b>	<b>Préprocessing des données</b>	<b>5</b>
<b>5</b>	<b>Choix du modèle teacher</b>	<b>6</b>
<b>6</b>	<b>Finnetunning du modèle teacher</b>	<b>6</b>
6.1	Motivation . . . . .	6
6.2	Approche . . . . .	6
6.3	Résultat . . . . .	7
6.3.1	Dataset n°1 . . . . .	7
6.3.2	Dataset n°2 . . . . .	8
6.4	Inférence . . . . .	8
6.5	Analyse . . . . .	10
<b>7</b>	<b>Implémentation ADD</b>	<b>11</b>
7.1	Structure générale du code . . . . .	11
7.1.1	Modèles . . . . .	11
7.1.2	Entrainement . . . . .	11
7.2	Analyse comparative de l'implémentation par rapport au papier ADD . . . . .	12
7.2.1	Points de fidélité par rapport au papier ADD . . . . .	12
7.2.2	Différences et limitations . . . . .	12
7.3	Résultats Expérimentaux . . . . .	13
7.4	Inférence . . . . .	15
7.5	Analyse . . . . .	16
<b>8</b>	<b>Conclusion</b>	<b>17</b>
<b>9</b>	<b>Annexe</b>	<b>18</b>

# 1 Introduction

Les modèles de diffusion sont récemment devenus incontournables dans le domaine de la génération d'images de haute qualité. Des travaux comme Stable Diffusion, DALL·E ou SDXL ont montré qu'il est possible de synthétiser des images très réalistes à partir de simples prompts textuels, grâce à un processus itératif de débruitage.

Cependant, l'échantillonnage (ou sampling) dans les modèles de diffusion classiques requiert de nombreuses étapes pour parvenir à une image nette. Typiquement, des dizaines ou des centaines d'itérations de débruitage peuvent être nécessaires, ce qui limite l'usage en temps réel. De leur côté, les GANs (Generative Adversarial Networks) [?] permettent, dans leur formulation standard, de produire une image de manière *single-step*, c'est-à-dire en une seule passe avant. Toutefois, malgré des progrès substantiels, la qualité et la diversité des GANs ne rivalisent pas toujours avec celles des modèles de diffusion *foundation* récents, et l'entraînement de GANs à grande échelle reste délicat (instabilités, effondrement de mode, etc.).

Adversarial Diffusion Distillation (ADD) se veut une approche de compromis. L'idée est de conserver la structure d'un modèle de diffusion (qui autorise plusieurs passages de débruitage progressifs), mais de distiller ce modèle en un générateur efficace, utilisable en très peu d'étapes (de 1 à 4). La stratégie centrale d'ADD est d'intégrer un *loss* adversarial (comme dans un GAN) au *loss* de distillation issu du modèle de diffusion maître (ou *teacher*). Ainsi, ADD cherche à combiner la fidélité des GANs en peu d'étapes et la qualité globale des grands modèles de diffusion.

Dans ce rapport, nous présentons une étude pratique de l'approche Adversarial Diffusion Distillation (ADD) appliquée à un jeu de données spécialisé dans les jeux vidéo Pokémons. Notre objectif est de comparer les performances de trois configurations distinctes :

1. Le modèle *teacher*, basé sur Stable Diffusion v1-4, utilisé sans *fine-tuning* (configuration de base).
2. Le modèle *teacher* après un *fine-tuning* classique, effectué sur deux *datasets* distincts de tailles différentes.
3. Le modèle *student*, entraîné via la méthode ADD (combinant distillation et apprentissage adversariale), également sur ces deux *datasets* de tailles différentes.

Cette étude vise à évaluer l'efficacité de la distillation adversariale (ADD) ainsi que l'impact du *fine-tuning* classique, en mesurant leurs effets sur la qualité des images générées et la rapidité d'échantillonnage, dans le contexte spécifique d'un *dataset* Pokémons relativement spécialisé. De plus, nous analysons l'influence de la taille des *datasets* sur les performances des modèles, afin de mieux comprendre les limites et les avantages de chaque configuration.

## 2 L'approche ADD

L'idée d'*Adversarial Diffusion Distillation* (ADD) est de prendre un *modèle de diffusion pré-entraîné (teacher)*, et de l'utiliser pour *distiller* ses connaissances dans un modèle *student*, en introduisant un terme de perte adversariale qui force le *student* à générer directement des images *réalistes à chaque étape*.

Ce faisant, le *student* est capable de faire très peu d'étapes de débruitage (1 à 4 étapes), tout en conservant la qualité des grands modèles. L'architecture d'ADD repose sur :

- **Un modèle student**, un U-Net ou équivalent, initialisé à partir d'un U-Net de diffusion existant,
- **Un discriminateur** léger, qui apprend à distinguer les images générées des vraies images,
- **Un modèle teacher**, fixé, qui fournit un *signal de distillation*.

L'entraînement combine deux objectifs : (1) *loss adversarial*, (2) *loss de distillation par score* (issu du *teacher*).

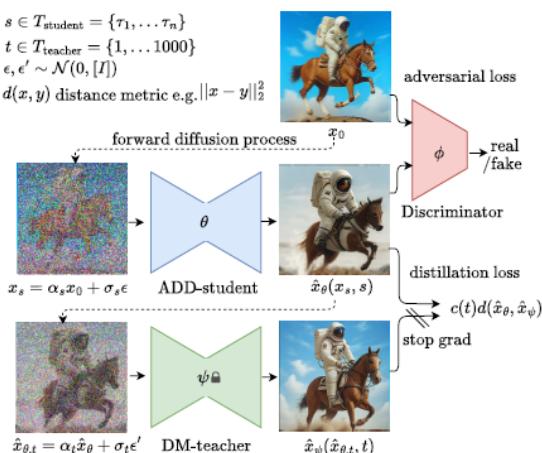


FIGURE 1 – Schéma représentatif de la méthode Adversarial Diffusion Distillation.

### 2.1 Formulation mathématique

Notons :

- $x_0$  une image réelle,
- $x_s$  l'image bruitée au *timestep*  $s$  (selon le planning de bruitage du *student*),
- $\hat{x}_\theta(x_s, s)$  la prédiction de débruitage du *student*,
- $\hat{x}_\psi(\cdot)$  la prédiction du *teacher* (censée être de haute qualité),

- $D_\phi$  le discriminateur,
- $\lambda$  le facteur de pondération entre les deux pertes.

**Bruitage forward (student).** Le *student* applique un *forward process* sur  $x_0$  pour obtenir  $x_s = \alpha_s x_0 + \sigma_s \epsilon$ , avec  $\epsilon \sim \mathcal{N}(0, I)$ . On choisit un ensemble de *timesteps*  $T_{\text{student}} = \{\tau_1, \dots, \tau_N\}$ , typiquement  $N = 4$ , pour couvrir les niveaux de bruit. Noter que  $\tau_N$  vaut 1000 (ou un grand nombre) pour couvrir le cas bruit très élevé.

**Loss adversarial.** Le *student* produit  $\hat{x}_\theta(x_s, s)$  qu'on compare à de vraies images  $x_0$  via le discriminateur  $D_\phi$ . Le *loss hinge* (souvent employé) peut s'écrire :

$$\mathcal{L}_{\text{adv}}^G = -\mathbb{E}_{s, \epsilon, x_0} \left[ \sum_k D_{\phi, k}(F_k(\hat{x}_\theta(x_s, s))) \right],$$

où  $F$  est un *réseau de features* (par ex. un ViT pré-entraîné), et  $D_{\phi, k}$  sont des têtes discriminantes légères sur différents blocs de  $F$ . Le discriminateur quant à lui est entraîné par :

$$\begin{aligned} \mathcal{L}_{\text{adv}}^D = & \mathbb{E}_{x_0} \left[ \sum_k \max\{0, 1 - D_{\phi, k}(F_k(x_0))\} + \gamma \mathcal{R}_1(\phi) \right] \\ & + \mathbb{E}_{\hat{x}_\theta} \left[ \sum_k \max\{0, 1 + D_{\phi, k}(F_k(\hat{x}_\theta))\} \right] \end{aligned}$$

où  $\mathcal{R}_1$  est un terme de régularisation (p. ex. *gradient penalty*).

**Loss de distillation.** On souhaite également rapprocher les prédictions du *student* de celles du *teacher*. On peut pour cela employer la *Score Distillation Sampling* (SDS) [?], ou des variantes. Une formulation générique est :

$$\mathcal{L}_{\text{distill}} = \mathbb{E}_{t, \epsilon'} [c(t) d(\hat{x}_\theta(x_s, s), \hat{x}_\psi(\hat{x}_\theta, t))],$$

avec

$$\hat{x}_{\theta, t} = \alpha_t \hat{x}_\theta(x_s, s) + \sigma_t \epsilon',$$

pour diffuser la sortie du *student* au niveau de bruit  $t$  propre au *teacher*. Ici,  $c(t)$  est un coefficient de pondération (parfois  $c(t) = \alpha_t$ , ou bien la pondération SDS), et  $d$  est une distance (typiquement  $\|\cdot\|^2$ ).

La combinaison donne l'objectif ADD :

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{adv}}^G(\theta, \phi) + \lambda \mathcal{L}_{\text{distill}}(\theta, \psi).$$

Le discriminateur  $D_\phi$  est optimisé par  $\mathcal{L}_{\text{adv}}^D(\theta, \phi)$ . Le *teacher*, lui, est **fixe** (*pas* réentraîné).

## 2.2 Stratégie d’entraînement

1. **Initialisation du student :** On part d’un U-Net déjà pré-entraîné comme modèle de diffusion (p. ex. Stable Diffusion ou SDXL). Cela évite les instabilités rencontrées en partant de poids aléatoires.
2. **Création de minibatchs :** On échantillonne des images réelles  $x_0$  (et les prompts texte associés si c’est du T2I), on en bruit une fraction pour le *student*, et on forme la paire  $(\hat{x}_\theta(x_s, s), x_0)$  pour le discriminateur.
3. **Calcul de la loss de distillation :** On diffuse  $\hat{x}_\theta$  jusqu’à un bruit  $t$  plus ou moins élevé (celui du *teacher*), et on compare  $\hat{x}_\theta$  à la sortie du *teacher*  $\hat{x}_\psi$  (qui fait un débruitage d’une image bruitée).
4. **Boucle d’optimisation :** On met à jour  $\theta$  en minimisant  $\mathcal{L}(\theta)$ , et on entraîne le discriminateur  $D_\phi$  à distinguer  $x_0$  de  $\hat{x}_\theta$ .

## 2.3 Usage en inférence

Une fois entraîné, le *student* ADD peut générer une image *en une unique étape* à partir de  $x_s$  totalement bruité ( $s = \tau_N$ ), ou en quelques étapes ( $N \leq 4$ ). À chaque étape, on dispose d’un échantillon d’image qui – grâce au *loss* adversarial – est déjà *sur le manifold des images réalistes*, et – grâce

au *teacher* – reste aligné avec le prompt ou la sémantique voulue.

Notons que, dans la plupart des expériences, *une seule étape* d’inférence donne déjà un excellent résultat. Si toutefois l’utilisateur souhaite peaufiner, on peut itérer, en appliquant la routine de débruitage  $N$  fois (en rediffusant légèrement et redénoisant).

## 2.4 Résultats et analyses

Les expérimentations dans la littérature ADD montrent :

- **Une qualité élevée en 1 à 4 étapes**, surpassant d’autres méthodes d’accélération de diffusion ou d’autres approches de distillation.
- **Une fidélité et un alignement prompt supérieurs** aux approches GAN classiques grâce au guidage par le *teacher* diffusion.
- **Des performances parfois supérieures** aux modèles *teacher* (p. ex. SDXL) en seulement 4 étapes. L’ajout du terme adversarial semble booster le réalisme (textures plus nettes, etc.).
- **Un éventail de résolution** allant jusqu’à  $512 \times 512$  (voire plus), rendu possible par la robustesse de l’entraînement adversarial, combiné à des U-Nets pré-entraînés à large échelle.

## 3 Dataset

### 3.1 Images

Notre dataset couvre l’ensemble des Pokémon de la génération 1 à 8 (du #1 au #809). L’idée est d’offrir un maximum de variété pour chaque Pokémon (plans de vue différents, poses, sprites issus de jeux variés, illustrations, etc.) et de bénéficier ainsi d’une diversité visuelle suffisante pour un bon apprentissage. C’est ainsi que deux datasets ont été créés

#### Évolution du dataset :

- **Dataset n°1** : un jeu de 809 images (une par Pokémon). Les résultats d’expérimentation se sont avérés trop limités (manque de diversité, sous-apprentissage, etc.).
- **Dataset n°2** : pour enrichir le dataset, nous avons ajouté deux nouveaux jeux de données, ce qui a porté le total à 11 325 images, souvent agrémentées de variantes (recadrages, rotations, modifications légères de couleur).



FIGURE 2 – Quelques échantillons du dataset de base

### 3.2 Dictionnaires

Les images étaient nommées tantôt par numéro de Pokédex, tantôt par nom de Pokémon, ce qui compliquait la correspondance entre fichiers et individus ciblés. Pour y remédier, nous avons construit un dictionnaire JSON reliant chaque numéro à un nom et organisé tous les visuels dans des sous-dossiers portant ce nom normalisé.

Le dataset textuel s'appuie sur :

- Un fichier `pokemon.csv`, qui recense noms et types de chaque Pokémons,
- Trois fichiers `pokedex_(Update.*).csv` pour les informations relatives à l'espèce (catégorie, etc.),

Après harmonisation et suppression des doublons, chaque Pokémons reçoit finalement une brève description (principalement axée sur l'apparence) produite via l'API OpenAI. Le fichier Excel final `pokemon.xlsx` regroupe, pour chaque Pokémons, son nom, ses types, son espèce, un chemin vers ses images et cette courte description. Les fichiers `labels.csv` et `labels.xlsx` listent l'ensemble des visuels disponibles avec leur chemin et les métadonnées associées.

## 4 Préprocessing des données

Avant d'aborder l'entraînement de nos modèles (teacher ou student via ADD), nous appliquons une série de prétraitements (*preprocessing*) sur les images :

- **Redimensionnement** : Chaque visuel est systématiquement redimensionné (par ex.  $512 \times 512$  ou  $256 \times 256$ ), afin de s'aligner sur l'architecture U-Net visée et de simplifier le traitement en lot (*batch*).
- **Normalisation des canaux** : On convertit les images au format RGB et l'on applique une normalisation (souvent canal par canal, comme Mean = 0.5, Std = 0.5), de façon à rendre le bruitage plus stable et cohérent.
- **Augmentations supplémentaires (optionnelles)** : Dans certains cas, on peut intégrer du *random crop*, *random flip*, ou des légères modifications colorimétriques, pour renforcer la robustesse du modèle et limiter l'overfitting.
- **Filtrage / Nettoyage** : Enfin, on élimine les images dont la résolution, le format ou le contenu n'est pas exploitable (images corrompues, doublons exacts, etc.).

Après ce préprocessing, nous obtenons un ensemble d'images homogènes, accompagné de métadonnées textuelles (nom, types, brève description). C'est à partir de ces données finalisées que nous exécutons les étapes d'entraînement pour le *teacher* (ré-entraînement ou fine-tuning) et pour le *student* (distillation via ADD).

## 5 Choix du modèle teacher

Le modèle *teacher* utilisé dans ce projet est `CompVis/stable-diffusion-v1-4`.

Bien que les auteurs de la méthode *Adversarial Diffusion Distillation (ADD)* recommandent l'utilisation de modèles plus récents, tels que `stabilityai/stable-diffusion-2-1` ou `stabilityai/stable-diffusion-2-0` (`SDXL`), le choix de Stable Diffusion 1.4 s'explique par sa légèreté. Avec environ 860 millions de paramètres et un encodeur texte CLIP ViT-B/32, il est nettement moins exigeant en mémoire que SD 2.1 (890 M, CLIP ViT-H/14) ou SDXL ( $\sim 2,3$  milliards de paramètres, CLIP ViT-G/14). Cette compacité en fait une solution plus simple à implémenter et particulièrement bien adaptée au prototypage rapide sur des ressources limitées.

Modèle	Paramètres	Encodeur texte	VRAM Inférence	VRAM Fine-tuning
SD 1.4	$\sim 860$ M	CLIP ViT-B/32	$\sim 4\text{--}6$ GB	$\sim 8\text{--}10$ GB
SD 2.1	$\sim 890$ M	CLIP ViT-H/14	$\sim 6\text{--}8$ GB	$\sim 12\text{--}14$ GB
SDXL (base)	$\sim 2.3$ B	CLIP ViT-G/14	$\sim 10\text{--}12$ GB	$\sim 20\text{--}24$ GB

TABLE 1 – Comparaison des modèles de diffusion : taille, encodeur et exigences mémoire

## 6 Finnetunning du modèle teacher

### 6.1 Motivation

Dans le but de mettre en évidence l'efficacité de l'approche ADD (*Adversarial Diffusion Distillation*), nous avons effectué deux finetunings distincts sur le modèle Teacher pré-entraîné (`Stable Diffusion v1-4`) afin de comparer leurs performances avec celles de l'approche ADD. Le premier finetuning a été réalisé sur le dataset n°1, tandis que le second a été conduit sur le dataset n°2. Ces expérimentations permettent d'évaluer les améliorations apportées par ADD par rapport à une optimisation classique du modèle Teacher sur ces deux ensembles de données.

### 6.2 Approche

Le finetuning a été effectué sur le modèle teacher de diffusion, en utilisant un dataset personnalisé de 12 000 images de Pokémon accompagnées de descriptions textuelles. L'objectif était d'adapter le modèle pour générer des images de Pokémon fidèles au style officiel, en optimisant deux composants clés : le `UNet`, responsable de la prédiction du bruit, et le `CLIPTextModel`, qui encode les prompts textuels. L'autoencodeur variationnel (VAE) a été gelé pour préserver la qualité de la conversion entre l'espace pixel et l'espace latent. Le dataset a été divisé en ensembles d'entraînement (80 %, 9600 échantillons), de validation (10 %, 1200 échantillons) et de test (10 %, 1200 échantillons) à l'aide d'un split stratifié pour garantir une répartition équilibrée.

Le processus d'entraînement repose sur un `DDPMScheduler` pour ajouter du bruit aux latents selon un horaire de bruit, suivi d'une prédiction du bruit par le `UNet`. Une perte MSE est utilisée pour minimiser l'écart entre le bruit prédit et le bruit réel, optimisée via `AdamW` avec accumulation de gradients. L'entraînement s'étend sur plusieurs époques, avec une validation après chaque époque pour surveiller la convergence et éviter le surapprentissage. Une évaluation finale sur l'ensemble de test mesure la capacité du modèle à généraliser.

Les hyperparamètres utilisés pour le finetuning sont résumés dans le tableau suivant :

TABLE 2 – Hyperparamètres du finetuning

Paramètre	Valeur
Taille du batch	4
Taux d'apprentissage	$1 \times 10^{-5}$
Nombre d'époques	5
Étapes maximales d'entraînement	100
Pas d'accumulation de gradients	1
Poids de régularisation ( <i>weight decay</i> )	$1 \times 10^{-2}$
Bêtas de l'optimiseur AdamW	(0.9, 0.999)
Résolution des images	$512 \times 512$
Étapes d'inférence	50
<i>Guidance scale</i>	7.5
Précision	<b>fp16</b>

Cette configuration assure un équilibre entre la stabilité de l'entraînement et la capacité du modèle à s'adapter aux spécificités du dataset Pokéémon, tout en exploitant les avantages de la précision mixte (**fp16**) pour optimiser les performances sur GPU.

### 6.3 Résultat

#### 6.3.1 Dataset n°1

Epoch	Avg Train Loss	Validation Loss	Test Loss
1	0.0370	0.0395	–
2	0.0345	0.0343	–
3	0.0337	0.0342	–
4	0.0354	0.0333	–
5	0.0356	0.0381	–
<b>évaluation Testset</b>		<b>0.0405</b>	

TABLE 3 – Résultats du finetuning sur le dataset n°1

Les résultats du finetuning sur le dataset n°1 montrent une bonne stabilité. Les pertes d'entraînement diminuent légèrement de 0.0370 à 0.0356 sur 5 epochs, tandis que la validation oscille entre 0.0395 et 0.0333, atteignant un minimum à l'epoch 4. Cette proximité entre les pertes indique une absence de surapprentissage. La perte sur le test set (0.0405) est cohérente avec la validation, suggérant une bonne généralisation du modèle.

### 6.3.2 Dataset n°2

Epoch	Avg Train Loss	Validation Loss	Test Loss
1	0.0808	0.0837	—
2	0.0784	0.0796	—
3	0.0770	0.0758	—
4	0.0757	0.0760	—
5	0.0768	0.0754	—
<b>Final Evaluation on Test set</b>			<b>0.0741</b>

TABLE 4 – Résultats du finetuning sur le dataset n°2

Les résultats sur le dataset n°2 révèlent également une bonne stabilité, mais avec des pertes globalement plus élevées que sur le dataset n°1. Les pertes d’ entraînement passent de 0.0808 à 0.0768 sur 5 epochs, montrant une amélioration progressive. La validation diminue de 0.0837 à 0.0754, avec un minimum à l’ epoch 5 (0.0754), indiquant une convergence graduelle sans surapprentissage évident. La perte finale sur le test set (0.0741) est légèrement inférieure à la meilleure perte de validation, confirmant une bonne généralisation.

Ainsi, les deux datasets montrent une stabilité lors du finetuning, sans signes de surapprentissage. Cependant, le dataset n°1 présente des pertes significativement plus faibles (environ 0.0333–0.0405) comparées à celles du dataset n°2 (environ 0.0741–0.0837), suggérant que le modèle performe mieux sur le premier. Cette différence pourrait être due à une complexité moindre ou une meilleure qualité des données dans le dataset n°1. Les deux datasets atteignent une convergence rapide (epochs 3–4), mais le dataset n°2 montre une amélioration continue jusqu’à l’ epoch 5, contrairement au léger rebond observé sur le dataset n°1. Globalement, le modèle généralise bien sur les deux datasets, avec une performance supérieure sur le dataset n°1.

## 6.4 Inférence

Les inférences ont été effectuées en utilisant les paramètres suivants :

- **num\_inference\_steps = 50** : Ce paramètre indique le nombre de pas d’inférence réalisés pour générer chaque image. Un nombre de pas plus élevé permet un débruitage plus progressif et aboutit généralement à une meilleure qualité d’image, car chaque étape affine la sortie générée. Cependant, il augmente également le temps de calcul.
- **guidance\_scale = 7.5** : Ce paramètre contrôle l’ intensité de la guidance (souvent appelée *classifier-free guidance*) pendant l’ inférence. Une valeur élevée renforce l’ adhérence du modèle aux informations de conditionnement (par exemple, le prompt textuel), ce qui améliore la fidélité des images générées par rapport au contenu demandé. Toutefois, une guidance trop forte peut limiter la diversité des résultats.

En outre, une interface web permet de configurer les paramètres de génération d’images, tels que le prompt, le nombre d’ itérations, et le Guidance Scale. Un aperçu de cette interface est visible en annexe de ce rapport (voir Figure 5).

Prompt	Teacher Vanilla	Teacher Finetuned	Teacher Finetuned12k
<b>1. Pikachu</b> a cute Pikachu with red cheeks playing in a grassy field			
<b>2. Eevee</b> a fluffy Eevee sitting under a tree with big expression			
<b>3. Lapras</b> a Lapras swimming in a crystal clear lake			
<b>5. Squirtle</b> a happy Squirtle splashing in clear blue water			
<b>6. Dragonite</b> a Dragonite soaring through clouds with its small wings			
<b>4. Flareon</b> a Flareon with fluffy fur that looks like flames			

TABLE 5 – Comparaison entre les innférences obtenues du modèle Teacher en version vanilla, finetunée (dataset n°1) et finetunée12k (dataset n°2) pour 6 prompts



TABLE 6 – Comparison entre le Pokémons original Giratina et les images générées par différents modèles.

## 6.5 Analyse

Après analyse des inférences, le modèle finetuné sur le dataset n°1 montre une amélioration par rapport au modèle vanilla. Le modèle de base génère des Pokémons de manière subjective, avec des postures et compositions souvent éloignées des attentes, manquant de cohérence avec le style officiel. En revanche, le finetuning, réalisé sur un dataset extrait des jeux Pokémons, produit des images nettement plus fidèles à l'esthétique originale. Les Pokémons générés affichent une posture centrée, avec une légère inclinaison vers la gauche ou la droite, respectant ainsi la structure et le style des personnages officiels, ce qui démontre une bonne assimilation des caractéristiques visuelles clés.

Pour le dataset n°2, le modèle finetuné génère des images de bien meilleure qualité que le modèle vanilla, avec une ressemblance accrue aux Pokémons de référence. Les Pokémons produits présentent plus de détails, une bonne forme générale, et une fidélité notable aux designs originaux. Le modèle a efficacement appris les features générales, telles que la luminosité, les couleurs, les lignes directrices et la structure globale, ce qui se traduit par des images visuellement cohérentes. Cependant, les résultats manquent de profondeur, et le modèle n'a pas pleinement assimilé des features plus abstraites, comme les nuances subtiles de texture ou les éléments contextuels complexes, limitant ainsi l'expressivité des générations par rapport à un rendu parfaitement réaliste.

Ainsi, Les deux modèles finetunés surpassent le modèle vanilla, produisant des Pokémons beaucoup plus fidèles aux styles officiels. Sur le dataset n°1, l'accent est mis sur une posture et une structure précises, avec une forte cohérence stylistique. Sur le dataset n°2, les images gagnent en détails et en qualité visuelle, mais souffrent encore d'un manque de profondeur et d'une capture limitée des features abstraites. Dans les deux cas, le finetuning améliore significativement la qualité, mais des ajustements supplémentaires pourraient être nécessaires pour capturer les nuances abstraites, notamment pour le dataset n°2.

## 7 Implémentation ADD

L’implémentation, telle que fournie, repose sur deux fichiers :

- `model.py` qui définit les classes nécessaires au bon fonctionnement du modèle `FrozenViTFeatureExtractor`, `ADDDiscriminator` et `PokemonADD`.
- `train.py` qui gère le *training loop*, incluant la lecture des données (ex. dataset Pokémons), la construction du noise scheduler, et la définition des pertes adversariales et de distillation.

Dans ce qui suit, nous rappelons brièvement la structure, puis nous discutons la correspondance entre cette implémentation et les éléments théoriques du papier ADD.

### 7.1 Structure générale du code

#### 7.1.1 Modèles

1. **Student U-Net** : Le code utilise un `UNet2DConditionModel` issu du dépôt `stable-diffusion-v1-4`. Il est chargé dans `PokemonADD` en tant que `self.student`. Il s’agit bien du réseau générateur que l’on désire entraîner par distillation + adversarial.
2. **Teacher U-Net** : Également un `UNet2DConditionModel` issu du même checkpoint, mais dont les poids sont gelés (`requires_grad = False`) pour guider le Student via la *score distillation*.
3. **Discriminateur** : Géré par la classe `ADDDiscriminator`. Il repose sur un extracteur de features ViT (c’est-à-dire `FrozenViTFeatureExtractor`), qui est gelé, et plusieurs “têtes” MLP (`n_heads = 3`) pour produire des logits adversariaux.
4. **Text encoder et VAE** : On observe que la classe `PokemonADD` charge un `CLIPTextModel` et un `AutoencoderKL` de stable diffusion, afin de gérer le conditionnement texte et l’encodage/décodage entre l’espace latent et les images en pixel space. Cette partie est standard dans la famille Stable Diffusion.

#### 7.1.2 Entrainement

Pour chaque batch, le code effectue globalement les opérations suivantes :

1. **Encodage texte.** Il récupère `text_embeds` (sequence embeddings) et `pooled` (vector embedding) via le `CLIPTextModel`.
2. **Bruitage partiel de l’image réelle.** L’utilisateur choisit un `partial_noise_level`  $\in \{0.0, 0.25, 0.5, 1.0\}$ , puis mélange  $\alpha_s = \sqrt{\alpha_{t_{int}}}$  et  $\sigma_s = \sqrt{1 - \alpha_{t_{int}}}$ . Cela produit un latent bruité  $\mathbf{x}_s$ .
3. **Passage Student.** Le Student (U-Net) prédit un latent de sortie.
4. **“Rediffusion” (Teacher).** La sortie du Student est re-bruitée à un `distill_noise_level` (fixé à 1.0 dans le script) puis traitée par le Teacher, ce qui donne la cible pour la perte de distillation.
5. **Loss adversariale.** La sortie du Student est décodée en pixel space, puis soumise au discriminateur (qui voit soit l’image générée, soit la vraie image). Cela crée un terme de *hinge loss* pour la partie discriminateur et un terme pour le générateur (Student).
6. **Loss de distillation.** L’utilisateur calcule un MSE entre la sortie Student et la sortie Teacher (en latents ou en pixel). Ce terme est ensuite combiné à la perte adversariale pour former la perte totale du Student.

## 7.2 Analyse comparative de l'implémentation par rapport au papier ADD

### 7.2.1 Points de fidélité par rapport au papier ADD

L'implémentation suit fidèlement plusieurs aspects clés du papier ADD, garantissant une conformité avec ses principes fondamentaux :

- **Combinaison adversarial + distillation.** L'approche centrale du papier, qui couple un discriminateur pour une fidélité en une passe et un signal de distillation basé sur un modèle Teacher, est respectée. Le discriminateur (`SDDDiscriminator`) évalue l'image décodée depuis les latents du Student, tandis qu'une perte MSE est calculée par rapport à la sortie du Teacher. La perte globale est définie par :

$$\mathcal{L} = \mathcal{L}_{adv} + \lambda \mathcal{L}_{distill}.$$

- **Initialisation depuis un modèle pré-entraîné.** Conformément au papier, le modèle Student est initialisé avec les poids d'un U-Net pré-entraîné (par exemple, Stable Diffusion), évitant un entraînement à partir de zéro et assurant une stabilité initiale.
- **Entraînement adversarial.** Un extracteur de features gelé basé sur ViT (`FrozenViTFeatureExtractor`) est utilisé avec des têtes MLP légères pour le discriminateur, suivant les recommandations du papier. La hinge-loss, parfois complétée par une régularisation R1, est également implémentée.
- **Unique teacher step.** L'implémentation adopte un unique passage par le Teacher après un re-bruitage de la sortie du Student, en ligne avec l'une des configurations proposées par le papier.
- **Flexibilité de la distillation.** La possibilité de calculer la perte de distillation dans l'espace latent ou pixel est intégrée via l'argument `use_pixel_dist`, offrant une flexibilité conforme aux options décrites.

### 7.2.2 Différences et limitations

Malgré une forte adhérence aux principes du papier, certaines divergences et limitations sont notables :

- **Horaire de bruit.** Le papier préconise un ensemble discret de timesteps (par exemple,  $\{1000, 500, 0\}$ ) pour entraîner le Student. L'implémentation utilise en revanche un `partial_noise_level` continu (converti en indices de 0 à 999), ce qui offre plus de flexibilité mais s'écarte de la grille fixe recommandée.
- **Pondération de la distillation.** Le papier explore des stratégies complexes de pondération (par exemple, exponentielle ou SDS/NFSD) en fonction du bruit  $\sigma_t$ . Ici, une pondération simplifiée (constante ou basée sur `alpha_s`) est adoptée, ce qui pourrait limiter la gestion des problèmes comme le "blurry teacher".
- **Passes multiples.** Le papier suggère d'entraîner le Student sur plusieurs étapes de débruitage consécutives pour améliorer la cohérence. L'implémentation se limite à un unique timestep par batch, sans enchaînement explicite de passes, ce qui ne tire pas pleinement parti de cette stratégie.
- **Conditionnement du discriminateur.** Le papier recommande d'injecter une image partiellement bruitée dans le discriminateur à faibles niveaux de bruit. L'implémentation utilise un embedding ViT sur l'image non bruitée, ce qui reste cohérent mais omet des optimisations comme la projection conditionnelle ou le dropout.

- **Ajustements dynamiques.** Le papier évoque des ajustements dynamiques d’hyperparamètres (comme  $\lambda$ ) ou des variantes comme NFSD. Dans l’implémentation, ces paramètres (`lambda_adv`, `r1_reg`) sont fixes, et des explorations comme l’emploi de LoRA ou l’impact de la taille du ViT ne sont pas abordées.

En résumé, l’implémentation capture l’essence de l’approche ADD, notamment en termes de combinaison adversarial-distillation, d’initialisation pré-entraînée et de structure du discriminateur. Toutefois, des simplifications dans la gestion des timesteps, de la pondération et des passes multiples, ainsi que l’absence de certaines optimisations, limitent son alignement complet avec les subtilités du papier. Ces différences pourraient affecter la qualité des images générées ou la robustesse de l’entraînement, mais l’approche reste globalement fidèle et fonctionnelle.

### 7.3 Résultats Expérimentaux

Dans cette section, nous présentons une analyse globale des cinq expériences menées sur le dataset n°1 afin d’évaluer l’impact de différents réglages sur l’entraînement du Student en mode ADD. Les paramètres modifiés incluent le coefficient  $\lambda_{adv}$ , la régularisation R1 (`r1_reg`), le taux d’apprentissage (`lr`), le mode de distillation (pixel vs. latents) ainsi que les niveaux de bruit partiels (`partial_noise_levels`).

Expérience	$\lambda_{adv}$	<code>r1_reg</code>	<code>lr</code>	Partial Noise Levels	Distillation
Exp1	1.5	$1 \times 10^{-6}$	$5 \times 10^{-6}$	{0.0, 0.25, 0.5, 1.0}	Latents
Exp2	3.0	$1 \times 10^{-5}$	$4 \times 10^{-6}$	{0.0, 0.25, 0.5, 1.0}	Latents
Exp3	0.3	$1 \times 10^{-6}$	$6 \times 10^{-6}$	{0.0, 0.25, 0.5, 1.0}	Latents
Exp4	1.0	$1 \times 10^{-5}$	$5 \times 10^{-6}$	{0.0, 0.25, 0.5, 1.0}	Pixel
Exp5	1.0	$1 \times 10^{-5}$	$5 \times 10^{-6}$	{0.0, 0.2, 0.4, 0.6, 0.8, 1.0}	Latents

TABLE 7 – Réglages des différentes expériences.

Expérience	Epoch 1				Epoch 5			
	d_loss	g_loss	distill	adv	d_loss	g_loss	distill	adv
Exp1	1.7983	0.8736	0.6282	0.2454	0.0731	2.3643	0.5226	1.1956
Exp2	1.8771	0.9421	0.7323	0.0782	0.1261	3.3324	0.5842	1.0687
Exp3	1.7707	0.7132	0.6715	0.0909	0.1225	0.8991	0.5992	1.3147
Exp4	1.6320	0.3672	0.1309	0.1163	0.0073	1.3452	0.1324	1.2231
Exp5	1.6693	0.7252	0.6302	0.0950	0.1486	1.6177	0.4363	1.2647

TABLE 8 – Valeurs des métriques pour Epoch 1 et Epoch 5 de chaque expérience

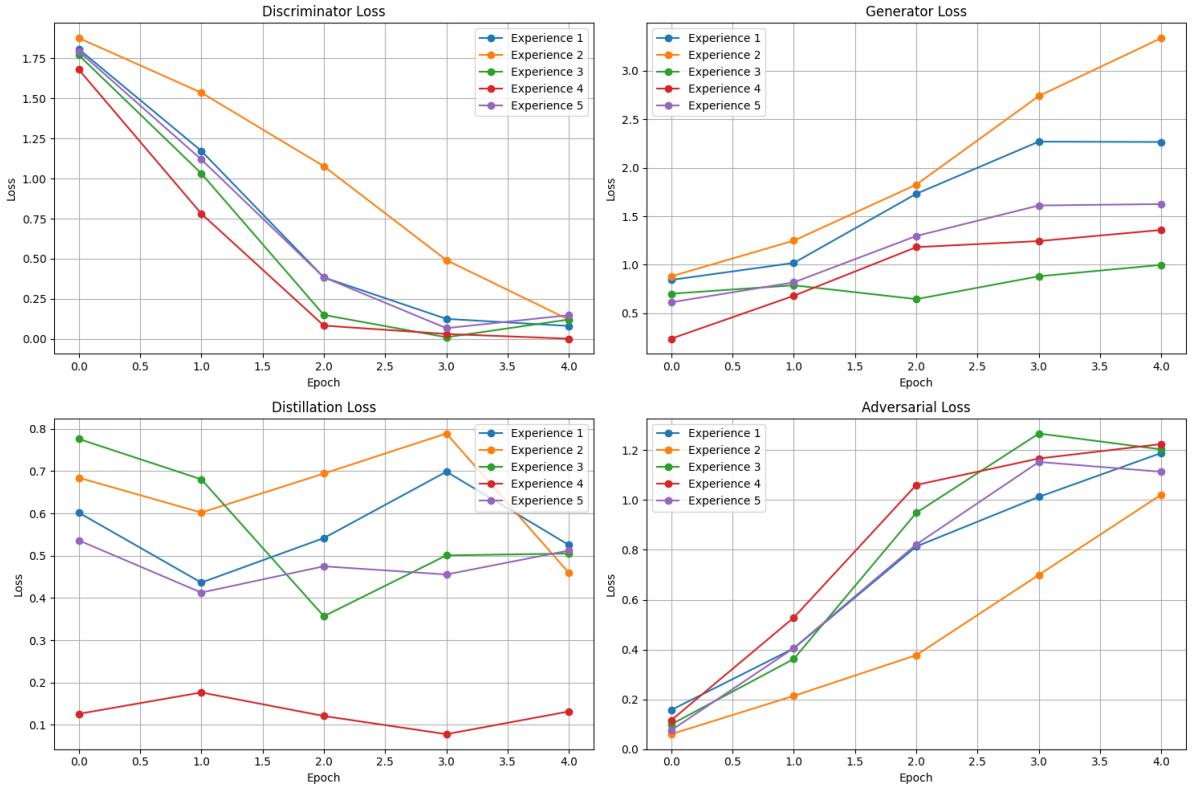


FIGURE 3 – Comparaison des mesures de perte entre cinq expériences et différentes époques. Les sous-graphiques affichent (en haut à gauche) la perte du discriminateur, (en haut à droite) la perte du générateur, (en bas à gauche) la perte par distillation et (en bas à droite) la perte antagoniste pour les expériences 1 à 5.

Sur le dataset 1, on observe une convergence rapide du discriminateur, dont la perte chute fortement dès les premières étapes. En parallèle, la perte du générateur augmente progressivement, traduisant une compétition efficace entre les deux réseaux. La perte de distillation reste instable, alternant entre valeurs faibles et pics élevés, ce qui reflète des difficultés ponctuelles du Student à reproduire fidèlement les sorties du Teacher. Enfin, la perte adversariale croît de manière régulière, signe que l’entraînement progresse vers un équilibre adversarial, bien que certaines fluctuations suggèrent encore un manque de stabilité sur ce dataset de taille réduite.

Afin d’améliorer les performances obtenues dans le cadre de l’expérience n°1, celle-ci a également été reproduite sur le dataset n°2, qui contient un volume d’images bien plus conséquent que le dataset n°1. Cette démarche vise à analyser l’impact de la taille du jeu de données sur l’efficacité de l’entraînement du Student dans l’approche d’Adversarial Diffusion Distillation (ADD). L’objectif est de vérifier si une plus grande diversité d’images permet au modèle d’apprendre des représentations plus robustes et cohérentes, améliorant ainsi la fidélité des images générées par rapport à celles du Teacher.

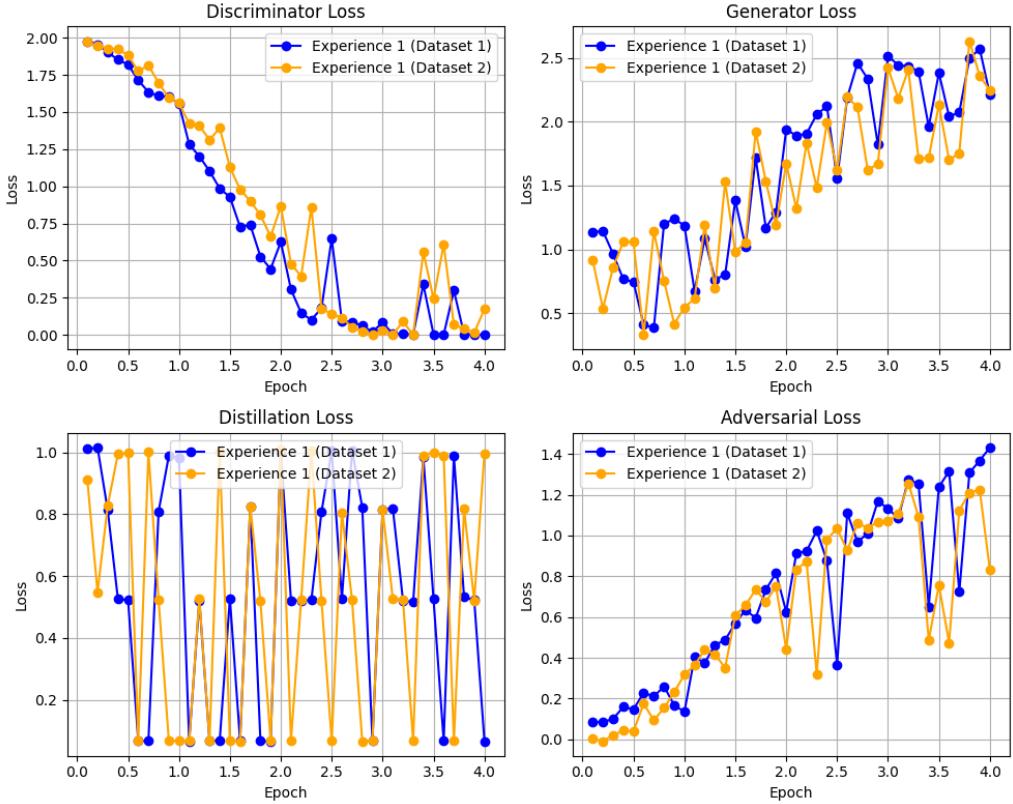


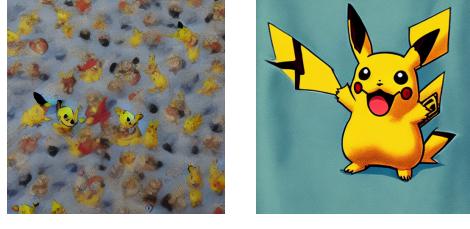
FIGURE 4 – Comparaison des différentes pertes (Discriminator, Generator, Distillation et Adversarial) pour l’Expérience 1 réalisée sur les datasets 1 et 2

Les courbes de pertes montrent des dynamiques d’apprentissage globalement similaires entre les deux datasets. Le discriminateur converge dans les deux cas, avec une descente légèrement plus rapide sur le dataset 1. La perte du générateur suit une tendance croissante comparable, bien que légèrement plus stable sur le dataset 2. Les pertes de distillation et adversariale présentent des variations similaires, avec quelques différences mineures de régularité. Ainsi, l’augmentation de la taille du dataset n’a pas entraîné de différence majeure en termes de performance, mais semble contribuer à une très légère amélioration de la stabilité globale de l’entraînement.

#### 7.4 Inférence



TABLE 9 – Résultats générés par les modèles des différentes expériences numérotées sur le dataset n°1. Le prompt utilisé est "un Pikachu mignon" avec un nombre d’étape d’inférence fixé à 4)



Step 4

Step 40

TABLE 10 – Images générées par le modèle de l’Expérience 1 (Dataset 2) au pas d’inférence 4 et 40 pour le prompt "un Pikachu mignon".

## 7.5 Analyse

Dans l’ensemble, toutes les expériences montrent une tendance commune : le discriminateur finit par être dominé (saturation de la  $d\_loss$ ) et le Student se repose sur le signal adversarial pour s’éloigner du Teacher. En effet, après plusieurs epochs d’entraînement, il apparaît clairement que le modèle Student ne parvient pas à s’adapter correctement aux données fournies, notamment lors des inférences en mode 4-step. Alors que le modèle Teacher continue de générer des images fidèles et de haute qualité, respectant la structure et la précision attendues, le Student présente des dégradations significatives : les inférences sont moins cohérentes, avec des compositions erronées et une perte notable des détails, ce qui se traduit par des résultats bien pires. Ce constat suggère que la stratégie de distillation et la configuration actuelle des hyperparamètres ne permettent pas au Student de capturer adéquatement les caractéristiques essentielles du dataset. Puisque la performance finale dépend de l’équilibre entre la perte de distillation et le signal adversarial, il est nécessaire de réévaluer et d’ajuster le schéma d’entraînement afin d’assurer une meilleure convergence du Student, capable de rivaliser avec le Teacher dans un cadre d’inférence multi-step.

En outre, les performances sont très faibles aux premiers pas d’inférence, avec des images floues, peu nettes et dépourvues de structure cohérente. Ce n’est qu’à mesure que le nombre de steps augmente que la qualité visuelle s’améliore progressivement. Or, l’objectif même de l’approche Adversarial Diffusion Distillation est précisément d’obtenir de bonnes performances dès les premiers steps. Ces observations suggèrent que, malgré les variantes testées, l’implémentation proposée reste insuffisante pour atteindre cet objectif.

Pour aller plus loin, il est recommandé d’augmenter le nombre d’epochs, de pratiquer une data augmentation plus poussée pour renforcer le discriminateur et d’expérimenter avec le paramétrage de  $r1\_reg$  (par exemple, l’augmenter à  $1 \times 10^{-4}$  si besoin).

## 8 Conclusion

C'est ainsi que le finetuning du modèle *teacher* sur deux datasets distincts a démontré son efficacité, notamment grâce à un second dataset plus riche qui permet de mieux capturer la structure, les détails et la diversité attendus. Ce résultat souligne l'importance d'un corpus d'entraînement conséquent et diversifié pour adapter des modèles pré-entraînés à des contextes spécialisés. Les modèles fine-tunés permettent aussi bien de reproduire un Pokémon existant que d'en générer un totalement inédit à partir d'une simple description textuelle.

En revanche, l'efficacité de l'approche *ADD* n'a pas pu être mise en évidence de manière convaincante. L'approche Adversarial Diffusion Distillation, quoique prometteuse, reste délicate à mettre en place. Nos premiers essais, malgré une adhésion aux principes fondamentaux (perte de distillation + perte adversariale), ont mis en évidence la difficulté d'équilibrer les forces en jeu : un *student* qui peine à reproduire la qualité du *teacher* lorsque le niveau de bruit et le nombre d'étapes de débruitage sont réduits. L'interaction entre la pondération de la distillation, la régularisation du discriminateur et le taux d'apprentissage doit être ajustée finement pour assurer la convergence et préserver la qualité de génération.

En outre, des perspectives d'amélioration claires se dégagent. D'une part, il serait pertinent d'explorer l'utilisation d'un modèle *teacher* plus performant, mieux aligné avec l'état de l'art, et de prolonger le finetuning pour renforcer l'adaptation au domaine spécialisé. Par ailleurs, des expérimentations comparatives en gelant un plus grand nombre de couches, ou en ajustant finement les stratégies d'optimisation, pourraient apporter des gains supplémentaires. En ce qui concerne l'approche ADD, une meilleure implémentation couplée à une recherche approfondie des hyperparamètres (taux d'apprentissage, régularisation, scheduling de bruit, etc.) est nécessaire pour tenter de démontrer pleinement son potentiel, en combinant la rapidité des méthodes adversariales avec la qualité des modèles de diffusion de nouvelle génération.

## 9 Annexe

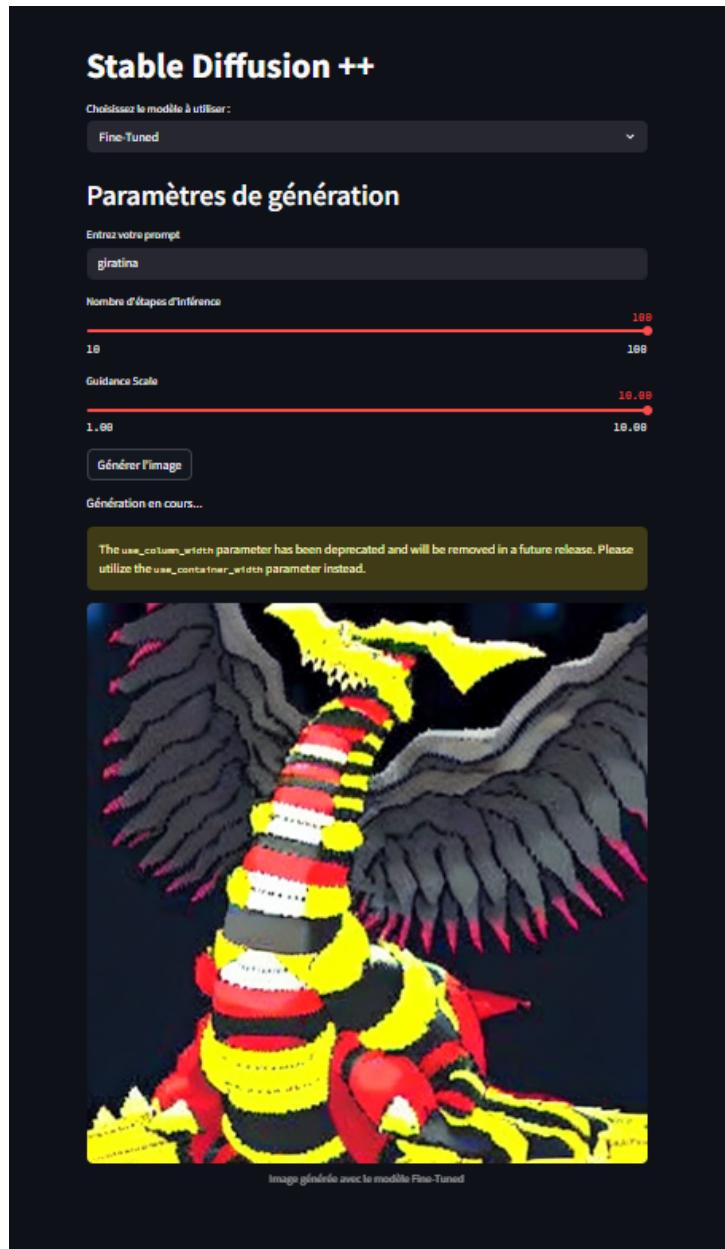


FIGURE 5 – Screenshot de l'interface web de Stable Diffusion ++ avec un des modèle affiné (Fine-Tuned)

## Références

- [1] A. Sauer, D. Lorenz, A. Blattmann, R. Rombach. *Adversarial Diffusion Distillation*. Stability AI, 2023. <https://arxiv.org/abs/2311.17042>
- [2] Datasets Pokémon Originels : <https://www.kaggle.com/datasets/hlrhegemony/pokemon-image-dataset> <https://www.kaggle.com/datasets/kvpratama/pokemon-images-dataset> <https://www.kaggle.com/datasets/vishalsubbiah/pokemon-images-and-types> <https://www.kaggle.com/datasets/mariotormo/complete-pokemon-dataset-updated-090420>
- [3] Poids du modèle teacher : *Stable Diffusion v1-4*, CompVis. <https://huggingface.co/CompVis/stable-diffusion-v1-4>
- [4] Poids du modèle ViT : *ViT small patch16 224 (DINO)*. [https://huggingface.co/timm/vit\\_small\\_patch16\\_224.dino](https://huggingface.co/timm/vit_small_patch16_224.dino)
- [5] Code : <https://github.com/BillyH20/add0>