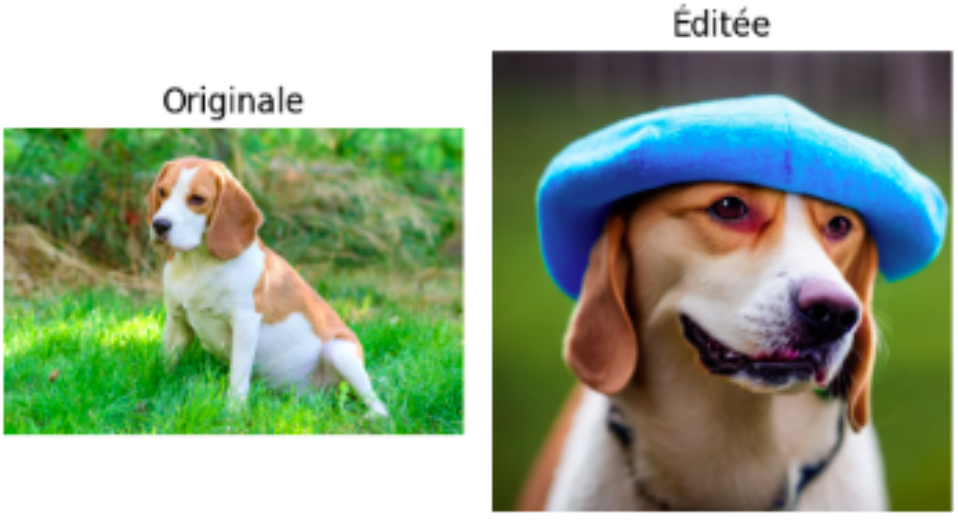
Gen-AI Report

12/ 04 / 2025

Robin Meneust, Maxime Cabrit, Antoine Charlet, Ethan Pinto

Imagic: Text-Based Real Image Editing with Diffusion Models



Summary

Introduction

Dans le cadre de notre projet d'IA Générative, nous avons décidé de reproduire le modèle Imagic décrit dans le papier Imagic: Text-Based Real Image Editing with Diffusion Models par Bahjat Kawar et al. Notre travail inclut également une modification visant à améliorer les performances du modèle. Le modèle Imagic consiste à modifier, grâce à un prompt, une image tout en gardant une grande partie des caractéristiques de l'image d'entrée.

Outil utilisées

Pour réaliser ce projet, nous avons mobilisé les outils et technologies suivants :

- **Langage et Environnement** : Python 3.12, exploité via des notebooks Jupyter ou des scripts dédiés. L'accélération matérielle a été assurée par CUDA pour les calculs sur GPU Nvidia. La gestion de version a été effectuée avec Git.

- **Modèle de Diffusion Fondamental** : Stable Diffusion v1.4

Nous avons basé notre implémentation sur le modèle pré-entraîné Stable Diffusion v1.4. Il s'agit d'un modèle de type Latent Diffusion Model (LDM), reconnu pour sa capacité à générer des images de haute qualité à partir de descriptions textuelles. Son architecture repose sur plusieurs composants clés que nous avons exploités et adaptés :

- Variational Autoencoder (VAE) : Ce composant encode les images de l'espace pixel (haute dimension) vers un espace latent (basse dimension) plus compact, et décode les représentations latentes en images pixels.

- UNet : Le cœur du processus de débruitage. C'est un réseau neuronal convolutif avec une architecture encodeur-décodeur et des connexions résiduelles (skip connections). Conditionné par le prompt textuel et le niveau de bruit (timestep), il apprend à prédire le bruit ajouté à une représentation latente pour l'inverser progressivement.

- Encodeur de Texte : Basé sur CLIP (ViT-L/14), il transforme le prompt textuel d'entrée (par exemple, e_{tgt}) en un embedding numérique que le UNet peut comprendre pour guider la génération d'image. C'est ce composant qui fournit e_{tgt}.

- Scheduler : Gère l'algorithme de diffusion/débruitage. Il définit comment le bruit est ajouté pendant l'entraînement et comment l'image est progressivement débruitée pendant l'inférence sur un nombre défini de timesteps.

L'interaction de ces composants permet à Stable Diffusion de générer des images cohérentes avec un prompt textuel. Notre travail avec Imagic a consisté à optimiser l'embedding textuel et à fine-tuner spécifiquement le UNet de ce modèle pré-entraîné pour l'édition d'images existantes.

- **Bibliothèques Principales** : diffusers (Hugging Face) pour l'accès et la manipulation aisée des modèles de diffusion, PyTorch pour les opérations tensorielles et l'entraînement des réseaux, transformers pour l'encodeur de texte.

Le modèle

L'implémentation du modèle Imagic décrit dans le papier scientifique nécessite l'implémentation de trois étapes que nous présentons dans cette section.

1. Optimisation de l'embedding

L'objectif de cette première étape est de trouver un embedding (e_{opt}) qui, une fois fournit au modèle de diffusion pré-entraîné, permet de reconstruire au mieux l'image d'entrée.

Pour cela, on génère tout d'abord un embedding correspondant au prompt (e_{tgt}). Ensuite, en fixant les poids du modèle de diffusion, on optimise cet embedding, c'est-à-dire qu'on modifie l'embedding et non les poids du modèle.

Pour ce faire, on prend l'image d'origine, à laquelle on ajoute du bruit grâce au scheduler, puis on l'envoie dans le modèle de diffusion qui va prédire le bruit qui a été ajouté à l'image. Ensuite, on calcule la loss entre le bruit prédit et le bruit initial. Cette loss est ensuite utilisée pour mettre à jour les valeurs de l'embedding pour l'optimiser et obtenir une prédiction de plus en plus fiable. Nous avons exécuté cette boucle 1000 fois avec un learning rate de 2e-3.

Le résultat de cette optimisation est un nouvel embedding, noté e_{opt}, ajusté pour générer une image ressemblant fortement à l'image d'entrée.

2. Fine tuning du modèle de diffusion

Une fois l'embedding optimisé (e_{opt}), la deuxième étape consiste à fine-tuner le modèle de diffusion, plus précisément le modèle Unet. Bien que l'embedding e_{opt} obtenu à l'étape 1 soit conçu pour que le modèle pré-entraîné génère une image similaire à l'originale, la reconstruction obtenue n'est souvent pas parfaite. Cela s'explique par le fait que le modèle de diffusion est généraliste. Comme il a été entraîné sur un vaste dataset, il peut manquer les détails fins ou textures spécifiques de l'image d'entrée. Ainsi, afin que le modèle de diffusion apprenne les caractéristiques spécifiques de l'image d'entrée et puisse la reconstruire correctement, il faut le fine-tuner. Pour cela, on fige l'embedding (e_{opt}) et on met à jour les poids du modèle.

Le processus de fine-tuning suit les étape classique d'entraînement d'un modèle de diffusion :

1. On prend l'image d'entrée originale.
2. On sélectionne un temps (timestep) t aléatoire.
3. Le modèle prédit le bruit à partir de l'embedding (e_{opt}) et de l'image bruité
4. On calcul la loss entre le bruit ajouté et le bruit prédit.
5. On met à jour les poids du modèle en fonction de la loss calculée.

Nous avons effectué cette boucle 1500 fois avec un learning rate très faible (5e-7) ce qui permet de ne pas trop modifier le modèle entraîné tout en s'assurant qu'il arrive à capter les subtilités de l'image d'origine.

Effectuer un fine tuning sur le modèle Unet est très important. En effet, si l'on ne réalise pas de fine tuning on obtient, pour l'embedding e_{opt}, une image qui ne ressemble pas totalement à l'image d'origine. Cela est problématique car le modèle n'arrivera pas à générer une image avec la modification souhaitée cohérente à l'image d'entrée.

3. Interpolation embedding

Cette étape est la dernière à devoir être réalisée avant l'inférence. Elle consiste à trouver un embedding final (ε) en réalisant une interpolation linéaire entre les deux embeddings obtenus précédemment (e_{opt} et e_{tgt}) :

$$\epsilon = \eta * e_{tgt} + (1 - \eta) * e_{opt}$$

où η est un hyperparamètre compris entre 0 et 1.

- Si $\eta = 0$, alors $\varepsilon = e_{opt}$. L'embedding utilisé est donc celui obtenu lors de l'optimisation de l'embedding c'est-à-dire celui optimisé pour reconstruire l'image d'origine. Comme on a fine-tuné notre modèle avec cet embedding on obtiendra une image en sortie (lors de l'inférence) quasiment similaire à celle en entrée.
- Si $\eta = 1$, alors $\varepsilon = e_{tgt}$. Alors l'embedding utilisé sera celui du texte cible. L'image générée (lors de l'inférence) correspondra donc à ce que le modèle produirait pour ce texte, sans tenir compte de l'image d'entrée.

Ainsi plus η est proche de 0, plus on aura une image fiable par rapport à l'image d'entrée mais ne prenant pas en compte ou très peu l'embedding (modification demandé demandé dans le prompt). Au contraire si on prend un η proche de 1, on obtiendra un image qui correspondra plus à l'embedding qu'à l'image d'origine.

4. Inférence

Après avoir optimisé e_{opt}, fine tuné notre modèle de diffusion et enfin calculé ε par interpolation entre e_{opt} et e_{tgt}, on peut effectuer l'inférence.

Pour cela, on commence par créer une représentation latente constituée de bruit aléatoire, selon une distribution Gaussienne ayant les dimensions attendues par notre modèle de diffusion. On envoie ensuite au modèle fine-tuné cette représentation latente ainsi que l'embedding interpolé. On configure ensuite un scheduler afin de lui spécifier le nombre d'itérations de débruitage a réaliser.

Ainsi pour chaque itération, le modèle reçoit l'état latent, l'embedding, et le timestep du scheduler.

A la fin, on obtient une représentation latente finale, qui est la version débruitée correspondant à l'embedding interpolé. Il suffit ensuite d'envoyer cette image dans le modèle VAE permettant de passer de cet espace latent à l'image pixélisée finale.

Améliorations apportées

Dans le cadre de ce projet, au-delà de la reproduction du modèle Imagic, nous avons exploré deux pistes d'amélioration ou d'adaptation potentielles :

1. Exploration de la Cross-Entropy Loss

Nous avons envisagé l'utilisation de la fonction de perte Cross-Entropy, typiquement employée dans les tâches de classification. L'idée était d'explorer si cette métrique, qui mesure la différence entre deux distributions de probabilités, pourrait offrir une alternative ou un complément à la perte MSE (Mean Squared Error) standard utilisée dans les modèles de diffusion pour comparer le bruit prédit et le bruit réel. L'hypothèse était qu'elle pourrait potentiellement mieux capturer certaines divergences lors de l'optimisation

de l'embedding (e_{opt}) ou du fine-tuning du modèle, bien que son application directe dans ce contexte ne soit pas standard et nécessiterait une adaptation spécifique.

2. Implémentation de LoRA (Low-Rank Adaptation)

Confrontés aux limitations matérielles (VRAM) lors du fine-tuning complet du modèle UNet de Stable Diffusion (étape 2 du modèle Imagic), nous avons implémenté LoRA. Il s'agit d'une technique de Parameter-Efficient Fine-Tuning (PEFT) qui permet d'adapter de grands modèles pré-entraînés avec un coût computationnel réduit. Au lieu de ré-entraîner tous les poids du modèle, LoRA gèle le modèle de base et injecte des paires de petites matrices "décomposées" (low-rank) dans certaines couches clés (typiquement les couches d'attention de l'UNet). Seules ces quelques matrices additionnelles sont entraînées. Cela réduit drastiquement le nombre de paramètres à optimiser (comme vu dans notre test, seulement ~0.046% du total), diminuant significativement l'usage VRAM et le temps d'entraînement, tout en visant à conserver une bonne performance d'adaptation pour la tâche spécifique (ici, apprendre les caractéristiques fines de l'image d'entrée).

Résultats obtenus

1. Approche Cross-Entropy

L'utilisation de la Cross-Entropy au lieu de MSE, n'a pas permis d'améliorer les résultats. En effet, cette approche n'a pas amélioré les résultats et s'est montrée moins performante pour la tâche d'édition d'image, comme l'illustre l'exemple du chien en première page. On observe que l'image générée avec la Cross-Entropy, bien que gardant une ressemblance, préserve moins les détails et la posture d'origine du chien que les exemples illustrés dans le papier. Nous concluons donc que la perte MSE reste la méthode la plus adaptée pour cette tâche.

2. Approche LoRA :

Nos premières tentatives d'entraînement d'un LoRA sur un dataset plus conséquent pour apprendre un concept spécifique ont échoué sur le matériel disponible en raison du temps de calcul excessif et des ressources mémoire encore importantes, bien que réduites par rapport à un fine-tuning complet.

Nous avons cependant réalisé une implémentation de démonstration (voir notebook Test_LoRA_implementation_Ethan.ipynb) pour valider la méthodologie sur un cas simplifié : apprendre au modèle Stable Diffusion 1.4 à générer une voiture jaune spécifique (associée au token unique "sks car") à partir d'une seule image d'exemple (car.jpg). L'entraînement a ciblé les matrices to_q et to_v des couches d'attention de l'UNet avec un rang (rank) de 4 pendant 500 étapes.

Le processus d'entraînement et d'inférence avec les poids LoRA chargés a fonctionné techniquement : le modèle a pu être entraîné sans crash sur ce cas minimaliste et l'inférence a produit une image influencée par le concept appris.

- Exemple de résultat (démonstration) :
- Prompt utilisé pour l'inférence : "photo of sks car driving on a beach"
- Image générée :

Bien que l'image générée ne soit pas parfaite (ce qui est attendu vu la limitation extrême des données d'entraînement à une seule image et un faible nombre d'étapes), ce test ne montre pas que la méthodologie LoRA est fonctionnelle en pratique. En effet, elle permettrait d'adapter le modèle de diffusion pour intégrer un nouveau concept de manière computationnellement plus abordable qu'un fine-tuning complet, mais dans les fait, des ressources plus importantes et un entraînement plus long sont nécessaires pour obtenir des résultats probants et de haute qualité.

Difficultés rencontrées

La réalisation de ce projet s'est heurtée à plusieurs défis majeurs :

1. Limitations des Ressources de Calcul :

L'implémentation d'Imagic, en particulier l'étape de fine-tuning du modèle de diffusion (UNet de Stable Diffusion), exige une quantité significative de mémoire GPU (VRAM). Les ressources disponibles (machines de l'école, instances Google Colab standard) se sont avérées insuffisantes, conduisant fréquemment à des erreurs "CUDA out of memory".

Cette contrainte a fortement limité notre capacité à expérimenter avec différents hyperparamètres ou à traiter des images de plus haute résolution. Pour contourner partiellement ce problème, une organisation spécifique a dû être mise en place où les tâches gourmandes en VRAM étaient exécutées séquentiellement par le membre de l'équipe disposant du matériel le plus performant, ralentissant le cycle de développement et de test itératif.

2. Contraintes Temporelles Générales :

Le projet s'est déroulé sur une période courte (environ un mois), qui coïncidait avec d'autres obligations académiques (cours, examens) et professionnelles (alternance, Projet de Fin d'Études).

Ce manque de temps dédié global a rendu difficile l'exploration approfondie de certaines pistes d'amélioration ou la résolution exhaustive des problèmes techniques rencontrés, notamment l'optimisation fine des hyperparamètres ou la mise en place de stratégies de debugging plus complexes.

3. Difficultés d'Organisation et de Collaboration Distribuée :

La présence d'un membre de l'équipe en séjour d'études à l'étranger a introduit des défis logistiques supplémentaires.

Le décalage horaire a considérablement réduit les créneaux de travail et de communication synchrones, complexifiant la coordination, la prise de décision collective et la résolution rapide de problèmes en équipe. La charge de cours de ce membre a également ajouté une pression sur les contraintes temporelles globales du projet.

Conclusion

Ce projet a permis de reproduire les étapes fondamentales du modèle Imagic (optimisation d'embedding, fine-tuning, interpolation) sur Stable Diffusion 1.4. Nous avons également exploré l'usage de la Cross-Entropy et implémenté LoRA pour adresser les limitations matérielles.

Les contraintes majeures de VRAM, de temps et d'organisation ont empêché d'atteindre des résultats d'édition satisfaisante. Néanmoins, nous avons validé notre compréhension et l'implémentation technique du pipeline Imagic. La démonstration LoRA, bien que basique, nous a permis de comprendre une méthode bien plus accessible de modification de modèle pour correspondre à un style ou à un ensemble de critère attendu pour le rendu des images générées.

Ce travail constitue une expérience d'apprentissage significative sur les modèles de diffusion, les techniques d'édition guidée par texte et les défis pratiques de l'IA générative.