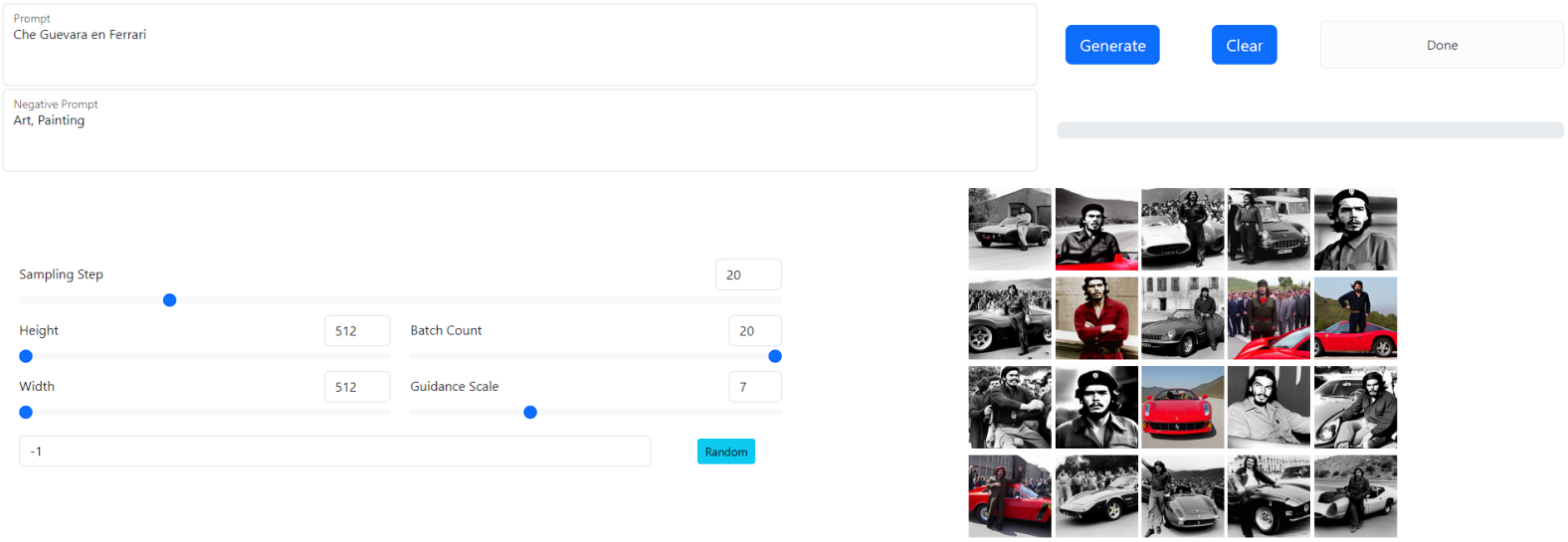
Ugo Maiurano
Gaëtan Coraboeuf

INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
UNIVERSITÉ D'ANGERS

Stable Diffusion
SAÉ S4



Tuteur : Philippe Lucidarme

4 boulevard Lavoisier, 49000 Angers

/!\ DISCLAIMER /!\

Ce rapport présente le travail réalisé par un groupe d'étudiants dans le cadre d'un projet pédagogique. Les auteurs et l'Université d'Angers ne garantissent pas que l'information, les documents, la méthodologie et le matériel présentés dans ce document soient complets, conformes à l'état de l'art et exacts ni n'assurent en toutes circonstances la sécurité des biens, des personnes et des utilisateurs. Les auteurs et l'Université d'Angers ne seront pas tenus responsables des dommages éventuels qui pourraient résulter de l'utilisation du contenu du présent rapport.

Sommaire :

Remerciements

- L'IUT d'angers
- Notre tuteur : Philippe Lucidarme
- M.Leduc
- Le groupe de projet
- Le technicien informatique
- Les personnes du magasin de composants électroniques de l'IUT
- M.Mangeard

Glossaire

IA (Intelligence Artificielle) : Un domaine de l'informatique qui se concentre sur la création de systèmes capables de réaliser des tâches qui nécessitent normalement l'intelligence humaine.

Modèle : Un modèle dans le contexte de la génération d'images est une représentation mathématique ou informatique d'un système qui peut générer des images. Il est pré-entraîné sur un ensemble de données existantes pour apprendre les motifs et les structures des images et peut ensuite générer de nouvelles images similaires.

Prompt : Dans le contexte de la génération d'images, un prompt peut être une instruction donnée pour guider le modèle dans la création d'une image spécifique. Il peut s'agir d'une description textuelle ou d'un ensemble de conditions spécifiques fournies en entrée pour influencer le contenu ou le style de l'image générée.

Négatif Prompt : Un négatif prompt est une instruction ou une spécification donnée pour éviter certains résultats indésirables lors de la génération d'images. Par exemple, il peut être utilisé pour éviter la création d'images offensantes, inappropriées ou incohérentes avec les normes éthiques ou les critères définis.

Guidance Scale ou CFG Scale (Context-Free Grammar Scale) : est une métrique utilisée pour évaluer la qualité et la cohérence des images générées par les modèles. Elle mesure à quel point les images produites sont conformes aux caractéristiques visuelles, aux structures de composition et aux règles esthétiques attendues.

Sampler : Dans le contexte de la génération d'images, un sampler fait référence à un composant ou à un algorithme qui permet de générer des images à partir de distributions de probabilité. Il peut être utilisé pour sélectionner différentes parties d'une image ou pour contrôler des aspects spécifiques tels que la luminosité, la couleur ou le style.

Seed : Une seed (ou graine) dans le contexte de la génération d'images est une valeur initiale utilisée pour initialiser l'algorithme de génération. Elle permet de démarrer le processus de création d'images à partir d'un point de départ spécifique, ce qui peut influencer le résultat final.

Socket : Tunnel de communication entre deux machines via un port réseau, souvent utilisé pour dialoguer entre deux serveurs.

ETA (Estimated Time of Arrival) : Dans le contexte de la génération d'images, ETA est une estimation du temps prévu pour terminer la génération d'une image ou d'un ensemble d'images. Cela dépend de la complexité du modèle, des spécifications de la génération et des ressources informatiques disponibles pour exécuter le processus.

Web Socket : Sur-couche du Socket qui est dédié plus à la communication avec le serveur et le client.

Introduction

Au cours des dernières décennies, les avancées technologiques ont façonné notre monde de manière sans précédent, et l'intelligence artificielle est au cœur de cette révolution. L'IA a le potentiel d'impacter tous les aspects de notre vie quotidienne, de l'automatisation des tâches routinières à la résolution de problèmes complexes. Alors que nous avançons vers un avenir de plus en plus connecté, l'influence de l'IA ne cesse de s'étendre, promettant de révolutionner des domaines tels que la médecine, l'éducation, l'industrie et bien d'autres.

Parmi les exemples remarquables de l'essor de l'IA, on retrouve les systèmes de génération de texte tels que ChatGPT ou d'image comme DALL-E et MidJourney. Ces modèles ont fait forte impression auprès du public en démontrant leur capacité à générer du texte, des images et même des œuvres d'art réalistes. Leur utilisation dépasse largement les simples échanges de conversation et souligne le potentiel de l'IA pour stimuler la créativité et repousser les limites de la perception humaine.

Cependant, un défi majeur se pose dans cette ère de l'IA : la collecte de données massives. Les modèles d'IA actuels nécessitent des quantités considérables de données pour s'entraîner et fournir des résultats pertinents. Cette dépendance aux données pose des problèmes en termes de confidentialité et de sécurité des informations personnelles. Cela à déjà été prouvé par les différent code source confidentiel fuité par ChatGPT. En effet, quand un employé utilise un tel outil pour développer toute information donner à l'IA sert à l'entraîner et peut se retrouver redonner à un autre utilisateur ayant un problème similaire.

Face à ce défi, une question cruciale émerge : est-il préférable d'héberger les modèles d'IA localement, sur nos propres infrastructures ?

De cette question a démarré nôtre projet. Les deux exemples de génération d'image, Dall-E et MidJourney sont en source fermé mais des équivalents existent en open source. On arrive donc au modèle utilisé pour ce projet qui est Stable Diffusion. Cet outil est open source et les différents modèles sont entraînés par la communauté.

Contrairement a Dall-E et MidJourney qui proposent une belle interface utilisateur sous la forme d'une page web. Stable Diffusion fonctionne depuis un script python, cela réduit le nombre d'utilisateur potentiel et rend l'utilisation complexe puisqu'il faut éditer le fichier python pour modifier les paramètres de génération.

Nôtre projet est donc de simplifier l'utilisation de stable diffusion en se rapprochant de l'expérience fourni par Dall-E et MidJourney.

Cahier des charges

Nôtre cahier des charges à pour but principal de simplifier l'utilisation de stable diffusion. Pour atteindre ce but, plusieurs points nous ont été demandées :

- Avoir un interface web pour le client
- Les images sont générées sur une machine différent que celle du client
- Les paramètre de génération doivent être modifiables
- Gérer plusieurs utilisateurs simultanés

Livrables

Nos livrables étaient :

- Le serveur web
- Le front end avec un UI et UX agréable
- Le programme python pour générer les images
- Un Websocket et un socket pour dialoguer entre les éléments précédant

Choix technologiques

Serveur de génération d'image (Python) :

Ce serveur et le point de départ de la construction de ce projet, comme on l'a dit l'IA s'exécute avec un script Python (c'est juste pour l'exécution ce n'est pas le langage du code de l'IA), alors quoi de mieux qu'un serveur Python.

Ce serveur Python dialogue grâce à une Socket avec le serveur Web NodeJS.

Ce serveur se charge donc :
- De démarrer l'IA.
- De recevoir les informations de génération d'images du client.
- De mettre ces informations dans l'IA et lancer la génération des images.
- Et de renvoyer les images (bit à bit) avec leurs informations.

Serveur Web (NodeJS) :

On avait le choix de plusieurs langages de programmation comme le C que l'on connaît bien mais qui est peu adapté au Web. Le PHP qui lui est tout à fait adapté mais qui est en perte de vitesse. Et le NodeJS que l'on a choisi car en effet il est de plus en plus utilisé et il a l'avantage d'être facile d'utilisation car c'est en réalité du JavaScript qui a été adapté pour le côté serveur.

On l'a conçu pour qu'il fasse passerelle entre le client et le serveur qui s'occupe de la génération d'image. En effet pour ne pas surcharger le serveur python avec les requêtes des clients, on a dû séparer les deux serveurs.

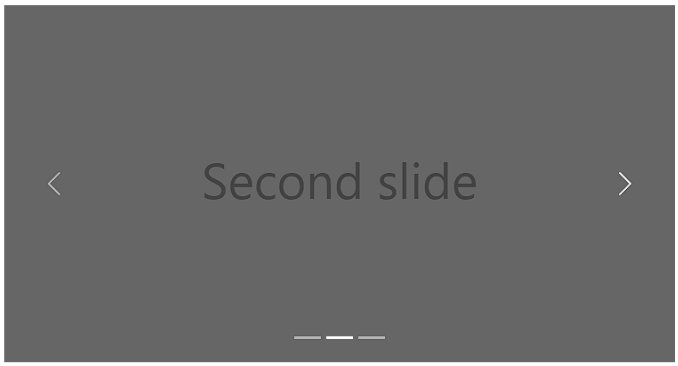
Ce serveur se charge donc :

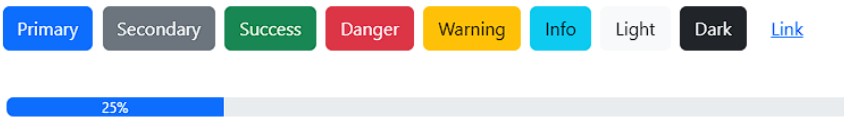
- De regarder si le serveur Python est opérationnel.
- D'envoyer la page Web au client.
- De récupérer les informations de génération d'images du client.
- De bufferiser les informations de génération d'images du client.
- D'informer le client de sa position dans le buffer de traitement.
- De renvoyer les informations quand le GPU du serveur Python est libre.
- De recevoir les images et les informations la consternante.
- Et de les renvoyer vers l'utilisateur qu'il l'a demandé.

Front End (Bootstrap) :

Pour simplifier et accélérer le développement du frontend, il a été choisi d'utiliser Bootstrap.

BootStrap est une librairie qui propose des blocs HTML déjà faits avec beaucoup de possibilités. Par exemple des boutons, des carrousels d'images ou des barres de progression





Cela réduit la tâche d'écriture de l'HTML et du CSS.

Cependant, utiliser une telle librairie a des limites, cela alourdit le client puisqu'il doit charger toutes la librairie en plus de la page. De plus, si les serveurs de Bootstrap sont indisponibles alors le client ne pourra pas charger la page.

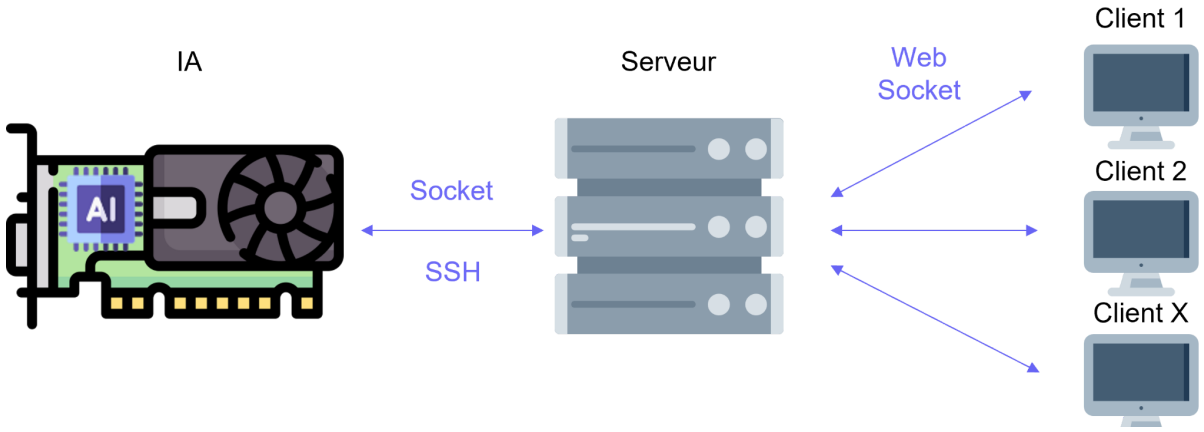
Même avec ces défauts la qualité de l'UI et l'UX est bien meilleure comparé à ce que l'on aurait développé sans Bootstrap avec la même contrainte de temps.

Front End (JavaScript) :

Le JavaScript est le troisième langage d'une page Web côté client, contrairement aux autres (HTML et CSS) qui sont plus pour le graphisme. Le JavaScript permet les intéractions de la page notamment avec le serveur et les liens avec les différents éléments graphiques.

Réalisation

Architecture :



L'architecture globale se présente comme ci-dessus. Les différents clients vont communiquer avec le serveur web via des Websocket. Le serveur va se connecter à la machine exécutant le code python en SSH puis les deux vont dialoguer avec une socket.

Socket :

Tunnel entre le serveur python et le serveur NodeJS.

Côté Python :

```python
import socket # Importer le module socket

# Créer un objet socket avec les paramètres par défaut IPv4 et TCP
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Se débarrasser du message d'erreur "Address already in use"
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

# Se connecter au serveur Node.js
s.connect((host, port))

s.send(message.encode())  # Envoyer un message encodé en bytes

# Recevoir les données du serveur
message_receive = s.recv(1024)  # Recevoir jusqu'à 1024 bytes

f.close() # permet de fermer la connection
```

Côté NodeJS :

```
import express from 'express'   // module express pour le serveur web
const app = express(); // Création du serveur web avec express

import http from 'http';    // module http pour le serveur web
const server = http.createServer(app); // Création du serveur http avec
express

import net from 'net'; // module net pour le serveur socket python
const pythonSocketServer = net.createServer(); // Création du serveur
socket python

// Démarrage du serveur socket python sur le port 3001
pythonSocketServer.listen(3001, () => {
 console.log('Python Socket Server started on http://localhost:3001');
// Affichage dans la console
});

pythonSocketServer.on('connection', (socket_python) => { /*code*/ });
// initialise la connection quand le serveur python se connecte

socket_python.on('data', (message) => { /*code*/ }); // Reçoit un
message de la part du serveur python (le code mit dans cette fonction
sera exécuté à chaque nouveaux message) (/!\ Le id du message est
forcément 'data')

socket_python.write(message); // Envoie un message au serveur python
```

L'un des problèmes avec les sockets c'est les id, on en a pas ce qui rend difficile de faire la différence entre deux messages (on ne sait pas à quoi correspond tel ou tel message).

Pour le résoudre un a décidé de mettre en place un protocole, on envoi "id:message", et à la réception on coupe au niveau de ":" pour tout récupérer.

Un autre problème était le passage de tableaux, car les sockets sont peu permissifs.

Pour le résoudre, on le convertit en chaîne de caractère pour l'envoyer, pour ce faire on utilise le JSON.

WebSocket :

Tunnel entre le client et le serveur NodeJS.

Côté client :

On inclut le dossier socket.io

```
var socket = io.connect(); // Déclare l'objet socket pour initialiser le WebSocket
socket.on('id', (message) => { /*code*/ }); // utilise l'objet socket initialisé avec le io.on pour récupérer le message qu'on nous a envoyé et donc l'id correspond.

socket.emit('id', message); // utilise l'objet socket initialisé avec le io.on pour envoyer un message avec un id.
```

Côté NodeJS :

```
import { Server } from 'socket.io'; // Importation du module socket.io
const io = new Server(server);   // Création du serveur socket.io

// Démarrage du Websocket sur le port 3000
server.listen(3000, () => {
  console.log('Server started on http://localhost:3000'); // Affichage dans la console
});

io.on('connection', (socket) => { /*code*/ }); // initialise la connection quand un client se connecte (le code mit dans cette fonction sera exécuté à chaque nouveaux client)

socket.on('id', (message) => { /*code*/ }); // utilise l'objet socket initialisé avec le io.on pour récupérer le message qu'on nous a envoyé et donc l'id correspond.

socket.emit('id', message); // utilise l'objet socket initialisé avec le io.on pour envoyer un message avec un id.
```

Manuel d’utilisation :

Prérequis :

Volta | Install volta

Installation :

Pour installer le projet, il existe deux solutions

- Depuis un navigateur | github.com/GaetanCoraboeuf/SAE_4

  Cliquez sur Code puis sur Download ZIP

- Depuis un terminal | ~ git clone https://github.com/GaetanCoraboeuf/SAE_4.git

  Nécessite d’installer git | ~ apt install git

Déplacez vous dans le fichier SAE_4/serveur/

Installer Node | ~ volta install node

Démarrage :

Déplacez vous dans le fichier SAE_4/

Pour démarrer le serveur plusieurs moyens s'offrent à vous.

Si le serveur web et l’IA sont sur la même machine

- Executer le script start.sh
- Un terminal est le serveur web, l’autre le python

Si le serveur web et l’IA ne sont pas sur la même machine

- Modifier le fichier start_IA.sh à la ligne 19 pour la connection ssh
- Executer le script start_IA.sh
- Un terminal est le serveur web, l’autre le python

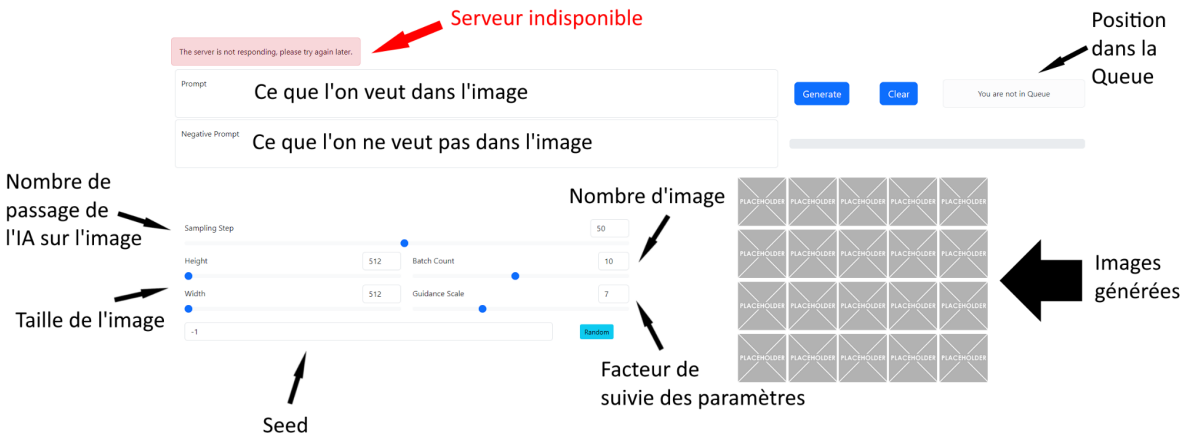
Si vous démarrer le serveur en ssh ou en wsl (Sans GUI, Konsole non utilisable)

- Modifier le fichier start_ssh_IA.sh à la ligne 19 pour la connection ssh
- Executer le script start_ssh_IA.sh
- Les terminaux sont fusionnées en un seule (serveur web en background)

Utilisation :

Lien de la page web : http://adresse_de_la_machine_hôte_du_serveur:3000

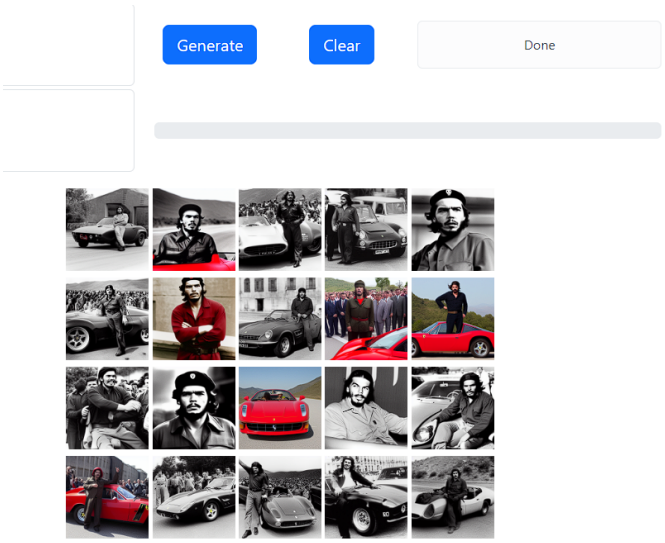
Agencement de l'interface :

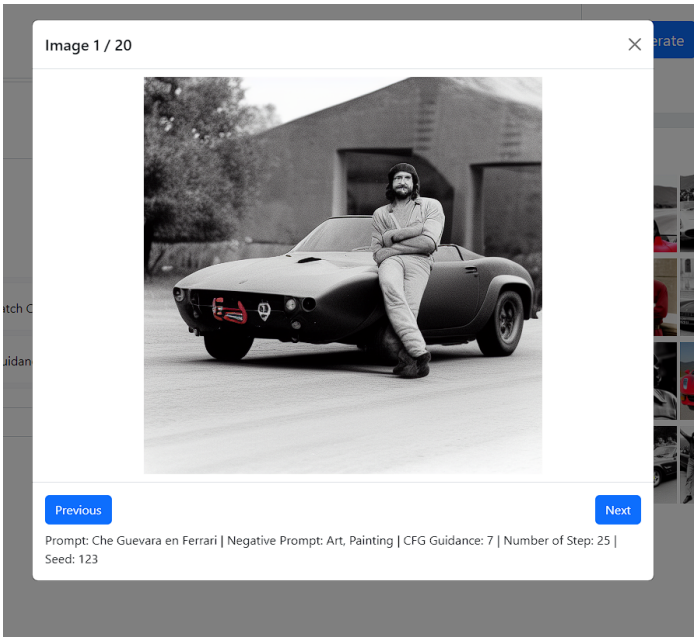


Après avoir choisis les paramètres, pour lancer la génération cliquer sur **Generate** ou appuyer sur **Entrer** dans le prompt.

Si il y a dans utilisateur avant vous la queue vous affichera votre position, elle précisera aussi quand ce sera vôtre tour et quand les images ont fini de vous être envoyées.

Les images sont affichées dans la grille, vous pouvez cliquer dessus pour les agrandir.





Le numéro de l'image s'affiche en haut à gauche. Vous pouvez naviguer entre les images avec les boutons Previous et Next. Les paramètres de génération sont affichés en dessous.

Conclusion

Le projet remplit le cahier des charges et il est utilisable. Néanmoins il reste quelques problèmes connus non résolus et d'amélioration possibles.

A certain moment l'utilisateur ne reçoit pas les images qu'il à demander mais cela reste rare et pourra être réparer rapidement

La génération des images est limitée par la puissance de la machine mais tout le reste du temps entre le clique de l'utilisateur et l'affichage des images pourrait être optimiser pour réduire le temps d'attente.

Perspectives

Le projet à encore de nombreuse évolutions et améliorations possibles :

- Le serveur peut avoir quelques problèmes lors de demandes simultanées ou successives.

  Différents problèmes peuvent arriver :

  - Les images d'autre utilisateur sont envoyées
  - Les images n'arrive jamais
  - La machine hébergeant le modèle est considéré comme occupé

- Avoir une bar de progression et un ETA (Estimated time of arrival) lors de la génération des images.
  Le front end est déjà en place, il reste à remonter l'information et la traiter.

- Permettre à l'utilisateur de choisir le sampler. Actuellement le sampler est Euler mais cela semble être modifiable dans le programme python.
  Il faut donc mettre en place une liste sur le front end et faire remonter l'information du choix jusqu'au python.

- Faire de la génération image vers image. Pour le moment seulement le texte vers image est utilisable mais des projets similaires possèdent cette fonctionnalité comme Stable Diffusion Web-UI, il serait donc possible de l'implémenter.

- Ne plus utiliser Bootstrap, le choix d'utiliser une telle librairie à été fait en connaissant les défauts mais cela à permis de grandement simplifier le développement du front end. Si de nouvelle fonctionnalité venait à être ajoutée, comme la précédente, il serait préférable de retirer Bootstrap.

Résumé

Français :

Ce projet permet de générer des images d'après un texte depuis votre propre machine. La machine génératrice et le serveur peuvent être différent. Les clients accèdent au service depuis une page web. En ouvrant le port 3000 vous pouvez accéder à votre serveur de n'importe où.

Anglais :

This project allows you to generate images from text on your own machine. The generator and the server can be on different computers. Customers access the service from a web page. By forwarding port 3000 you can access your server from anywhere.

Annexes

- Github du projet : https://github.com/GaetanCoraboeuf/SAE_4
- Installation Volta : Install volta
- Documentation Bootstrap : Doc Bootstrap 5.3
- Stable Diffusion WebUI : https://github.com/AUTOMATIC1111/stable-diffusion-webui