

Electronics and Computer Science  
Faculty of Engineering and Physical Sciences  
University of Southampton

Tomas Mrkva

May 2022

AI-assisted Generation of Artistic Images From Text  
Using Differentiable Drawing Rasteriser and CLIP

Project supervisor: Dr Jonathon Hare  
Second examiner: Dr Julian Rathke

A project report submitted for the award of  
MEng Computer Science

Word count: 9732



## Abstract

Generative models have received great popularity in computer vision and deep learning. Nowadays, it is possible to generate high-definition realistic images from text, complete unfinished pictures, or combine the content of images with various art styles.

In contrast, only a fraction of the work has been done in the vector image generation field. Due to their mathematical structure, vector images have the advantage of infinite scaling. They can also be useful for drawing/painting generation since the lines and curves can be used to represent strokes of pencil or brush. This approach was taken in *CLIPDraw*[1], a text-to-drawing algorithm that combined Open AI’s image classifier *CLIP*[2] with a differentiable renderer for vector graphics. The role of the renderer was to optimise drawings in a vector form and rasterise them so that they could be scored by *CLIP*.

This project aims to apply *CLIP* with another rasteriser from *Differentiable Drawing and Sketching*[3], perform experiments with the new algorithm, and compare the results with *CLIPDraw*. Furthermore, its objective is to use the algorithm with state-of-the-art natural language processing models to create an easy-to-use pipeline that can automatically produce illustrations from paragraphs or whole chapters of novels.

# Statement of Originality

- I have read and understood the [ECS Academic Integrity](#) information and the University's [Academic Integrity Guidance for Students](#).
- I am aware that failure to act in accordance with the [Regulations Governing Academic Integrity](#) may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

**I have acknowledged all sources, and identified any content taken from elsewhere.**

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

**I have used the following libraries:**

- **DifferentiableSketching:** (<https://github.com/jonhare/DifferentiableSketching>)
- **CLIPDraw:** (<https://github.com/kvfrans/clipdraw>)
- **Streamlit:** (<https://github.com/streamlit/streamlit>)
- **Hugging Face:** (<https://huggingface.co/>)
- **PyTorch:** (<https://pytorch.org/>)
- **Google Spreadsheets Python API v4:** (<https://github.com/burnash/gspread>)

**I did all the work myself, or with my allocated group, and have not helped anyone else.**

**The material in the report is genuine, and I have included all my data/code/designs.**

The complete database of the images used for comparison and experimentation is available in the archive and at:

<https://docs.google.com/spreadsheets/d/1JChM8Ssfqbv1QeDGvSIDMQXbXcmNupDjySvp1gWxbaM/edit?usp=sharing>

**I have not submitted any part of this work for another assessment.**

**My work did not involve human participants, their cells or data, or animals.**

*ECS Statement of Originality Template, updated August 2018, Alex Weddell [aiofficer@ecs.soton.ac.uk](mailto:aiofficer@ecs.soton.ac.uk)*

# Contents

|       |   |    |
|-------|---|----|
| 1     | List of Abbreviations .....   | 5  |
| 2     | Introduction.....   | 6  |
| 2.1   | Problem .....   | 6  |
| 2.2   | Goal .....  | 6  |
| 2.3   | Scope .....   | 6  |
| 3     | Literature review.....  | 7  |
| 3.1   | Introduction .....  | 7  |
| 3.2   | Variational Autoencoders (VAEs) .....   | 7  |
| 3.2.1 | DRAW: Deep Recurrent Attentive Writer.....  | 7  |
| 3.3   | Generative Adversarial Networks (GANs) .....  | 8  |
| 3.3.1 | Generative Adversarial Text to Image Synthesis .....  | 9  |
| 3.3.2 | StackGAN .....  | 9  |
| 3.4   | Autoregressive models .....   | 10 |
| 3.4.1 | PixelRNN and PixelCNN .....   | 10 |
| 3.4.2 | Transformer-based approaches .....  | 11 |
| 3.5   | Artistic approaches to image generation .....   | 12 |
| 3.6   | Raster images limitations .....   | 12 |
| 3.7   | Vector image generation .....   | 13 |
| 3.7.1 | Im2Vec.....   | 13 |
| 3.8   | CLIP .....  | 13 |
| 3.9   | CLIPDraw .....  | 14 |
| 3.10  | Differentiable Drawing and Sketching .....  | 14 |
| 3.11  | Conclusion .....  | 15 |
| 4     | Design and implementation of the new algorithm.....   | 16 |
| 4.1   | Design .....  | 16 |
| 4.2   | Implementation of the new text-to-drawing algorithm:<br>CLIP+DiffDrawing&Sketching (CLIP+DDS) ..... | 17 |
| 4.3   | Challenges faced during the implementation .....  | 18 |
| 5     | Experimentation.....  | 20 |
| 5.1   | Hyperparameters tuning .....  | 20 |
| 5.1.1 | Methodology .....   | 21 |
| 5.2   | Possible improvements to the algorithm .....  | 21 |
| 5.2.1 | Image augmentation.....   | 22 |
| 5.2.2 | Grey-colour initialisation.....   | 23 |
| 5.3   | The impact of a random seed value on the quality of the generated images ...                        | 24 |
| 6     | Comparison with CLIPDraw .....  | 26 |
| 6.1   | Methodology .....   | 26 |
| 6.1.1 | “Calibration” of the algorithms’ hyperparameters.....   | 27 |
| 6.2   | Evaluation of the results.....  | 28 |
| 6.2.1 | Resource efficiency.....  | 28 |
| 6.2.2 | Image quality .....   | 29 |
| 6.2.3 | Discussion of the results .....   | 31 |

|        |   |    |
|--------|---|----|
| 6.2.4  | Challenges faced .....  | 31 |
| 7      | Practical application of the algorithm .....                      | 32 |
| 7.1    | A high-level overview of the implementation.....                  | 32 |
| 7.2    | NLP models used .....   | 32 |
| 7.2.1  | Summarisation .....   | 32 |
| 7.2.2  | Headline/Keyword extraction .....                                 | 33 |
| 7.3    | User interface design and implementation .....                    | 34 |
| 7.4    | Deployment .....  | 34 |
| 7.5    | Testing.....  | 35 |
| 7.6    | Drawing completion.....   | 35 |
| 8      | Critical evaluation.....  | 37 |
| 9      | Conclusion and future work.....                                   | 38 |
| 10     | Personal reflection .....   | 39 |
| 10.1   | Project management .....  | 39 |
| 10.1.1 | Time management.....  | 39 |
| 10.1.2 | Risk management.....  | 39 |
| 11     | Bibliography .....  | 40 |
| 12     | Appendix.....   | 43 |
| 12.1   | Appendix A: Project Management .....                              | 43 |
| 12.2   | Appendix B: Risk Assessment .....                                 | 46 |
| 12.3   | Appendix C: Image augmentation techniques .....                   | 47 |
| 12.4   | Appendix D: UI Wireframes .....                                   | 49 |
| 12.5   | Appendix E: Screenshots of the <i>Streamlit</i> application ..... | 52 |
| 12.6   | Appendix F: Examples of the generated images using CLIP+DDS.....  | 55 |
| 12.6.1 | Database of the results .....                                     | 57 |
| 12.7   | Appendix G: Vector lines widening .....                           | 58 |
| 12.8   | Appendix H: GPU runtimes .....                                    | 59 |
| 12.9   | Appendix I: Original project brief .....                          | 60 |

# 1 List of Abbreviations

|             |                                      |
|-------------|--------------------------------------|
| <i>NLP</i>  | Natural language processing          |
| <i>SOTA</i> | State-of-the-art                     |
| <i>VRAM</i> | Video random access memory           |
| <i>UI</i>   | User interface                       |
| <i>GPU</i>  | Graphics Processing Unit             |
| <i>DDS</i>  | Differentiable Drawing and Sketching |

## 2 Introduction

Computer vision is a discipline of artificial intelligence which focuses on gathering meaningful information from images[4]. One of the many applications of this field is creating new images based on the insights gathered from vast labelled datasets during a learning process. This process is called image generation, and with the recent breakthroughs in modern machine learning approaches has evolved rapidly.

### 2.1 Problem

The primary metric in image generation has been photo-realism, sharpness or high-definition. In addition, most popular datasets used in computer vision only contained raster images (made of pixels on a raster grid) as they are widely used, compared to vector images (using mathematical equations to render the lines by a programme). Therefore, the generation of images in the vector format or artistic pictures that often do not aim to be entirely realistic has been overlooked.

### 2.2 Goal

The project has two goals.

The first is to research and reimplement the PyTorch[5] code of the CLIPDraw[1] text-to-drawing synthesis algorithm with the rasteriser proposed in [3], tune the hyperparameters and compare the quality of the results of the two algorithms.

The second goal is to develop an application that would use the modified algorithm to generate illustrations from the text. This is done using SOTA NLP models that can summarise large pieces of text and create headings that can be used as input to the modified drawing algorithm. Additionally, a simple UI is designed and developed for the pipeline's use.

### 2.3 Scope

The scope of this project is to build on top of the research done, mainly CLIP[2], CLIPDraw[1], Differentiable Drawing and Sketching[3] and create a new Text-to-Drawing synthesis algorithm. Tuning of the hyperparameters and exploring different techniques discussed in [1] is also expected. Any major modifications of [2] and [3] or the NLP models used for the application pipeline are outside the project's scope due to the time constraints and will not be explored. Furthermore, the complexity of the images used for generation and comparison is constrained by the available computing resources and time.



## 3 Literature review

### 3.1 Introduction

This literature review aims to analyse the main generative models in the context of image generation:

- (i) Variational Autoencoders
- (ii) Generative Adversarial Networks
- (iii) Autoregressive models

It aims to compare the results of their various adaptations and describe their advantages and shortcomings. It will highlight the limitations of current approaches to image generation in the context of rasterisation and describe one of the current SOTA generative vector models.

The text-to-image generation will be the primary focus; however, other techniques such as art generation will also be discussed. The limitations of current approaches to image generation in the context of rasterisation will be explained, and the motivation to implement a generative model that can work with vector graphics will be set. Finally, a new approach to both raster and vector image generation using a neural network *CLIP*[2] will be analysed, as it is the stepping stone to the methods used in this project.

### 3.2 Variational Autoencoders (VAEs)

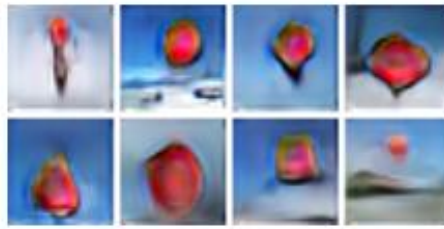
Variational encoders are based on the Autoencoder architecture. As described in [6], Autoencoders consist of the *encoder* and the *decoder*. The encoder compresses the data into a lower-dimensional “latent” space. The decoder does the reverse process: it tries to reconstruct the compressed data to get as close to the input data as possible.

The issue with vanilla autoencoders is that they *have a discrete representation of the original inputs in the latent space*[7]. Therefore, only random guesses can be made when generating new images, not producing good results[6]. Variational autoencoders resolve this issue as they represent the latent space as a continuous probability distribution[8]. This adjustment fixes the problem as the latent space is smooth, allowing sampling of data from the distribution.

#### 3.2.1 DRAW: Deep Recurrent Attentive Writer

One of the models that belong to the group VAEs is *DRAW: Deep Recurrent Attentive Writer*[9]. This model creates images in a human-like form as it generates parts of the scene independently from others, gradually refining the approximate sketches. Instead of generating the images in a single pass, the generation is done iteratively, accumulating the modifications throughout the process. To simulate human artists, which only focus on one area at a time, DRAW uses *attention gates* to focus only on selected areas at every step in the iteration while ignoring the rest. This is achieved by passing an image through a Gaussian filter centred around a focus area.

There are further extensions of this model, such as *alignDraw*[10] which after being trained on the Microsoft COCO dataset[11], can generate images from text describing scenes that the model has not seen from the dataset before, such as *A stop sign is flying in blue skies*[10] (*Figure 1*).




---

Figure 1: An example of a generated image using the alignDraw with the prompt: *A stop sign is flying in blue skies*. (Reproduced from[10] )

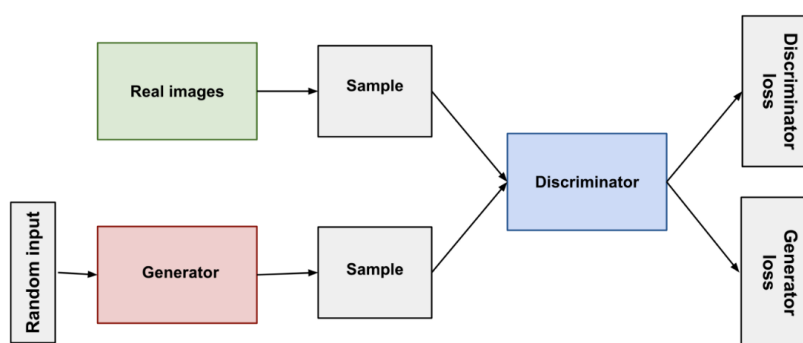
---

These models suffer from the major shortcoming of VAEs: the difficulty of generating fine-grained details. The created images are blurry. Objects with similar visuals, such as cats and dogs, are difficult to distinguish. This could be possibly due to the use of a small latent dimension while simultaneously trying to minimise the loss of the input data.

### 3.3 Generative Adversarial Networks (GANs)

GAN[12] is a generative model that was introduced in 2014. The network is based on a minimax algorithm where a generator  $G$  and a discriminator  $D$  compete. The generator model is responsible for generating new data that looks like it came out from the original dataset, trying to maximise the probability of  $D$  making a mistake in predicting whether the data comes from a real dataset or has been artificially generated. The discriminator's task is then to determine whether the data is real. The generator has no access to the real data and only learns through the discriminator's feedback. The training is done interchangeably for both generator and discriminator, pausing the training of the component that becomes better than the other, or stopping when the generated results become good enough[13].

This architecture (*Figure 2*) works very well for generating images, and its adaptations have been able to create realistic-looking pictures with fine-grained details.




---

Figure 2: A diagram of the GAN architecture during the training process. (Reproduced from [14])

---

### 3.3.1 Generative Adversarial Text to Image Synthesis

Reed et al.[15] used conditional GANs to generate images of flowers and birds based on text input, using the Caltech-UCSD Birds[16] and the Oxford-102 Flowers datasets. The model was, in many cases able to generate *visually plausible 64x64 images conditioned on text*[15]. Their approach was the *first end-to-end differentiable architecture from the character level to pixel level*[15]. Compared to alignDRAW[10], it generates *sharper and higher-resolution samples that roughly correspond to the query*.



---

Figure 3: Examples of the generated images from an unseen (zero-shot) text description: *this small bird has a pink breast and crown, and black primaries and secondaries*. (Reproduced from [15])

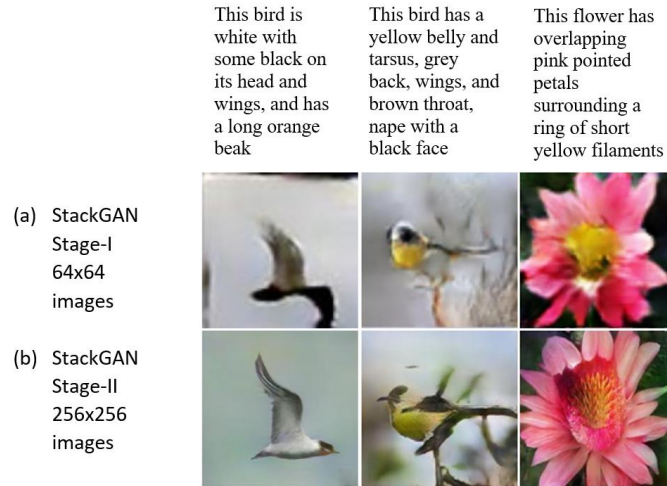
---

### 3.3.2 StackGAN

StackGAN[17] proposed a solution to improve the images' resolution, further enhancing the details in the generated images. Their model, which comprises 2 stacked conditional GANs, can generate photo-realistic images with a resolution of 256x256.

The Stage-I GAN generates basic shapes and colours of the object conditioned on the text; however, some details mentioned in the text might not be captured in the first stage.

The Stage-II GAN is built upon the results from Stage-I. It is conditioned on both the low-resolution image and the text embedding. This network corrects the defects in colour and completes the previously ignored details from the text, producing a higher-resolution image. StackGAN achieved a 28.47% improvement in terms of inception score on CUB dataset, and 20.30% improvement on Oxford-102[17] when compared to [15].




---

Figure 4: An example of the images produced by StackGAN from the text inputs above. The rough shapes and colours are generated in the first phase (a), while in the second phase (b), the sharpness and details are adjusted. (Modified from[17])

---

GAN-based models generally achieve better results than VAEs in the text-to-image synthesis, and more recent models can achieve high-quality or even photo-realistic images from text.

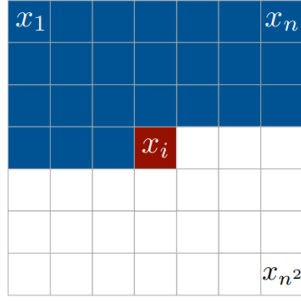
However, GANs have some significant drawbacks. One of the problems with GANs is that they can be challenging to train because they tend to get stuck in a poor local minimum and not reach convergence. A mode collapse, where the generator provides limited variety, is also possible, and there are doubts about whether GANs can learn the true data distribution[18]. Additionally, image-related limitations include difficulties with counting objects, distorted perspective, and problems with global structures such as the number of legs of an animal[13].

### 3.4 Autoregressive models

Autoregressive models are the third primary approach in image generation. The foundational research in this area was done with the proposal of *PixelRNN* and *PixelCNN* [19]. A significant improvement was achieved in [20] with the new architecture called the *Transformer*.

#### 3.4.1 PixelRNN and PixelCNN

Autoregressive models take the approach to *tractably model a joint distribution of the pixels in the image*[19] by expressing it as a product of conditional distributions. Therefore, the modelling problem is turned into a sequence problem, such that all the previously generated pixels determine the next pixel value (*Figure 5*).



$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, \dots, x_{i-1})$$

---

Figure 5 (left), Equation 1 (right): The probability of predicting an image made of  $n \times n$  pixels. The image  $\mathbf{x}$  can be expressed as a 1-dimensional vector  $x_1, \dots, x_{n^2}$ . The value  $p(x_i | x_1, \dots, x_{i-1})$  is the probability of  $i$ -th pixel given all the previous pixels from the start of the image to the  $i$  minus 1st pixel. The joint probability is the product of the conditional distributions over all pixels. (Reproduced from [19])

---

Recurrent neural networks, popular in NLP thanks to their ability to evaluate sequential information such as sentences using previously obtained information, can be adapted to this problem. Instead of sequences of words, pixel values are used.

PixelCNN, a modified model that used convolutional layers instead of recurrent ones, was much faster to train since it has the advantage of parallelisation but was found to produce worse results than its recurrent alternative.

The main advantage of PixelRNN and PixelCNN is the *tractability of the joint probability*  $p(\mathbf{x})$ [13], which makes it easier to track its performance. The training is also simpler than GANs since they do not have any previously discussed GAN-related risks. Due to their structure, these models are very good at completing images; however, they cannot generate as detailed images as GANs can.

### 3.4.2 Transformer-based approaches

In 2017, A. Vaswani et al.[20] proposed a new network architecture in the context of machine translation, the *Transformer*. Compared to RNNs, which have the constraint of being sequential, Transformers avoid recurrence and rely purely on the attention mechanism[20]. This allows them to be parallelised and therefore, be faster than RNNs while achieving SOTA results on translation tasks.

Even though they were primarily aimed at NLP-related tasks, transformers have performed very well in image generation too. OpenAI (<https://openai.com/>) released a model Image-GPT[21] which was able to generate complex image completions and whole images that contain *clearly recognisable objects*[22]. In 2021, they released DALL·E[23], a 12-billion parameter version of the GPT-3 transformer, trained to generate images from a text description using a dataset with text-image pairs. It received image pixels together with text as input and could autoregressively produce realistic-looking images.

These novel models can provide a much better quality of pictures than previous attempts and, in the case of DALL·E (*Figure 6*), can generate images with complex structures. However, due to their architecture, they need to be trained on massive datasets and require large amounts of computing power.



---

Figure 6: Images that the DALL·E model generated from the prompt: *an armchair in the shape of an avocado...* (Reproduced from [24])

---

### 3.5 Artistic approaches to image generation

The approaches to image generation discussed so far focused on realistic image generation. However, generative models can also be used to produce artistic images.

**CAN (Creative adversarial network)**[25] was built over GANs with certain modifications that make it perform better in generating creative images. In CAN, the generator receives two signals. The first signal is the output of the discriminator whether the previously generated image is “art or not art”. The second signal describes how well the discriminator can classify the art style from the art styles it was trained on. In experiments, CAN produced images that humans could not distinguish from paintings made by artists.

**Neural style transfer**[26] is a technique that uses two images as input: a content image and a style image (for example, an artwork) and combines them using Convolutional Neural Networks. The results look as if the content image was painted the same way as the style image.

**DeepDream**[27] is another approach that can be taken to generate artistic images called *activation maximisation*. It aimed to understand what happens in the different layers of artificial neural networks. An image can be fed to the network, letting the network analyse it. Then, a layer can be picked, and the network is asked to enhance whatever the layer has detected. Each layer deals with different features; for example, lower layers are mainly sensitive to edges, while higher layers deal with more sophisticated features or objects. This process can be applied iteratively, creating dream-like images.

### 3.6 Raster images limitations

All the previously highlighted techniques of image generation worked with raster images. Raster images are made of a grid, where each cell represents a pixel. The bigger the grid, the more resolution and detail the image has. However, these images are not scalable: when increasing the size of the image, it becomes blurry. Resampling techniques that can smooth out the grid exist, but they cannot entirely fix the blurriness.

Vector images are widely used in computer graphics areas such as graphic design. They are made of mathematical equations such as Bézier curves, lines, and points. The advantages of vector images are that they can be easily edited, resized without losing quality, and require less storage space. Their major drawback is that they are not suited for displaying realistic images such as photos, where the colours can change pixel by pixel.

Due to their advantages and drawbacks, vector images are well suited for generating artistic paintings and drawings since they are usually not as detailed as photos, and curves and lines can express hand strokes of a brush, better simulating the human way of painting. Nonetheless, only a few artificial generative models can create images in this format.

## 3.7 Vector image generation

### 3.7.1 Im2Vec

One of the problems with the unpopularity of vector image generators is the lack of training data in the vector format. The two leading vector-graphics generative algorithms, SVG-VAE[28] and DeepSVG[29] require vector graphics images for training.

Im2Vec[30] uses a differentiable rasteriser and differentiable compositing to work with vector graphics while internally learning on raster images. A variational encoder architecture is used to encode raster input image  $I$  into a latent code  $\mathbf{z}$ , which is then decoded into a set of ordered Bézier paths. These paths are then rasterised and composed. The obtained result, a rasterised image  $O$  can be compared to the target image. Since every step is differentiable, a loss between  $O$  and  $I$  can be computed, and error is backpropagated to the model using gradient descent, differentiating the output image with respect to Bézier parameters. A final image can be obtained in both rasterised and vector formats[27].

## 3.8 CLIP

CLIP[2] is an image classifier that can learn *image representations from natural language supervision*[2]. The main reason for this ability is that it was trained on 400 million diverse image/text pairs. These pairs were scraped from the internet and therefore did not only contain single words but rather descriptions in sentences. *CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples*[2]. This is done in a contrastive fashion: the cosine similarity between the correct image and the given text encodings is maximised, while the cosine similarity of incorrect pairings is minimised. As a result of this approach, CLIP is very flexible since it can match the performance of image classifiers without being trained on any specific datasets.

When used for zero-shot prediction, images and text are embedded by text and image encoders and projected into a latent space with the same dimension. Therefore, the dot product between the projected text and image is used to compute the similarity score.

There are some limitations as it has problems with tasks such as counting the number of objects in the image, classifying fine-grained details such as differences between car models, flower species, etc. In addition, it also performs poorly on images that are not well covered in its training dataset. Finally, it can only retrieve given captions and cannot generate new ones.



### 3.9 CLIPDraw

With the release of CLIP, a new approach to text-to-image synthesis can be taken. Instead of training a neural network, *synthesis through optimisation*[1] framework can be followed, where images are *generated through evaluation-time optimisation against a given metric*[1]. CLIPDraw[1] minimises the distance of CLIP encodings between the generated drawing and the text prompt given. Instead of the typical raster approach that uses a pixel matrix, it generates images by tweaking vector curves (RGBA Bézier curves).

CLIPDraw first generates an image with a specified number of random curves. The curves are then rendered to a pixel image using a differentiable rasteriser[31]. The raster image is augmented by perspective-shifting, cropping and resizing. This process is necessary as otherwise, the images *fulfil the numerical objective but are unrecognisable to humans*[1].

The batch of augmented images and the text prompt are then projected into the same latent space so a similarity score from CLIP can be obtained. The loss function is then computed using the cosine similarity between the two terms. This process forms the loss function. Since each part of the process is differentiable, gradient descent can be used to optimise the parameters of the curves, decreasing the loss every iteration and improving the quality of the image until the number of iterations reaches the target given (*Figure 7*).



---

Figure 7: Example of a single run of CLIPDraw algorithm with 250 iterations. After each iteration during the runtime, the parameters of each curve (position and colour) are tweaked s.t. the drawing corresponds to the given description: *A spaceship flying in a starry sky.* (Reproduced from [1]).

---

This process allows CLIPDraw to generate drawings in vector and raster form with different styles while matching the text description in often unexpected but interesting ways. An essential part of the optimisation process is done by the differential renderer from [31], which enables the vector drawings to be rasterised so they can be augmented and encoded by CLIP.

### 3.10 Differentiable Drawing and Sketching

While the rasteriser that used CLIPDraw used techniques developed in modern computer graphics, the differentiable rasteriser proposed in [3] took a bottom-up approach which *allowed for drawing operations to be composed in ways that can mimic the physical reality of drawing*...[3]. Differentiable relaxations (continuous approximations)[32] of a rasterisation process for points, lines and curves were explored so that their parameters could be optimised during gradient descent. In a 2D case, the parameters are the point coordinates, start and end coordinates of a line (*Equation 2*), or control points of Bézier curves.



$$f(\mathbf{n}; \mathbf{s}, \mathbf{e}) = \exp(-d_{seg}^2(\mathbf{n}, \mathbf{s}, \mathbf{e})/\sigma^2)$$

---

Equation 2: The equation of a differentiable relaxation of a line. The parameter  $\mathbf{n}$  corresponds to the pixel being rasterised,  $\mathbf{s}$  is the start coordinate, and  $\mathbf{e}$  is the end coordinate of the line.

$d_{seg}^2(\mathbf{n}, \mathbf{s}, \mathbf{e})$  is the square Euclidian Distance of a pixel  $\mathbf{n}$  from the line segment. This relaxation creates a scalar field (a heat map) where the value of each pixel corresponds to the distance of the closest point of the line. The parameter  $\sigma^2$  is used to control the width of the line. (Reproduced from [3])

---

Each segment is rasterised separately and then combined into one image using a differentiable composition function such as *softor*:

$$c_{softor}(\mathbf{I}^{(1)}, \mathbf{I}^{(2)}, \dots, \mathbf{I}^{(n)}) = \mathbf{1} - \prod_{i=1}^n (\mathbf{1} - \mathbf{I}^{(i)})$$

---

Equation 3: The softor function, each  $\mathbf{I}$  represents a segment defined in the image space. (Reproduced from [3])

---

This function works in the following way: if at any of the segments a pixel is “on”, the pixel will be “on” in the final image as well[3]. Additional methods, such as the *over* function used for coloured images, have been proposed.

The loss function can be represented as:

$$\min_{\boldsymbol{\theta}} \|R(\boldsymbol{\theta}) - \mathbf{T}\|_2^2$$

Where  $\mathbf{T}$  is the target image,  $R$  is a composition function, and  $\boldsymbol{\theta}$  is a vector storing all parameters. If the rasterisation function  $R$  is differentiable with respect to  $\boldsymbol{\theta}$ , the minimisation problem can be solved with gradient descent.

### 3.11 Conclusion

This review has covered different image generation approaches: VAEs, GANs and Autoregressive models. It highlighted some of their advantages and disadvantages, discussed their architectures on a high level and the performances they achieved. Additionally, alternative image-generation techniques such as CAN, Neural style transfer and DeepDream were explained. It is interesting to see how this field will evolve, especially with the recent success of autoregressive transformer-based approaches on large datasets. Furthermore, the advantages of vector-graphics image generation in the artistic domain were pointed out. Finally, with the CLIP’s release, a new method of text-to-image synthesis algorithms has become possible, and the approach taken in one of them, CLIPDraw, was analysed.

## 4 Design and implementation of the new algorithm

### 4.1 Design

The algorithm builds on top of research done in *CLIPDraw: a text-to-drawing synthesis through Language-Image Encoders*[1]. As CLIPDraw operates with vector strokes instead of pixels, it has two major advantages:

- (i) The generated images in vector format can be scaled without losing quality.
- (ii) The vector strokes such as lines and curves can be used to simulate the strokes of a brush and, therefore, better simulate the human way of painting.

In addition, since it was designed to use a pre-trained CLIP model during the optimisation process, no further training is required and the generation takes only a few minutes (depending on the number of vector strokes and iterations) to produce a result.

---

**Algorithm 1: CLIPDraw** (modified from: [1])

---

**Input:** Image description *desc*, Iteration Count *I*, Curve Count *N*, Augment Size *D*, pre-trained CLIP model

**Begin:**

```
EncDesc = CLIP(desc)           #Encode the image description desc.
Curves0..N = RandomCurve()    #Randomly initialize the curves.
#optimisation loop
for i = 0 to I do
    Pixels = DiffRender(Curves)    #Render Curves to Pixels.
    AugBatch0..D = Augment(Pixels) #Augment the Image.
    EncImg = CLIP(AugBatch)        #Encode the augmented images.
    Loss = -CosineSim(EncDesc, EncImg) #Compute Loss.
    Curves ← Minimize(Loss)        #Backpropagation.
```

**end for**

---

First, the text prompt is encoded with the CLIP text encoder into a tensor of size [1, 512], representing a 512-dimensional vector. *Tensors are mathematical objects that can be used to describe properties, just like scalars and vectors. They are merely a generalisation of scalars and vectors; a scalar is a zero rank tensor, and a vector is a first-rank tensor*[33]. Then, the position parameters and the colour channels of curves are randomly generated.

Throughout the optimisation loop, the CLIPDraw algorithm iteratively tweaks the parameters of RGBA Bézier curves. In every iteration, the curves are rendered into a raster image. Then, the augmentation of the image was performed. As the authors discovered, the augmentation of images is necessary to produce recognisable images *for humans*[1]. The augmentation techniques used in CLIPDraw were:

```
torch.transforms.RandomPerspective
torch.transforms.RandomResizedCrop.
```

Further augmentations were explored in the section [5.2.1](#).

Each augmented image is then encoded by the CLIP image encoder into a tensor of size [1, 512].

Since the CLIP encodings of the augmented images and CLIP encoding of the text description have the same dimensions, the *cosine similarity* (Equation 4) between the encodings of the augmented images and the encoding of the image description can be computed. Each step of the process is differentiable, and therefore, the backpropagation algorithm[34] (Equation 5) can be used to minimise the loss. The minimisation process is performed with an optimiser. Optimisers are used to update weights (in this case, the parameters of the lines) dynamically based on heuristics such as the gradient of the loss instead of a set learning rate. In CLIPDraw, the Adam[35] optimiser was used.

$$S_c(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

---

Equation 4: Cosine similarity between vectors  $\mathbf{A}$  and  $\mathbf{B}$

---

$$\frac{\partial L}{\partial \text{Curves}}$$

$$\text{where } L = S_c \left[ \text{CLIP} \left( \text{Augment}(\text{DiffRender}(\text{Curves})) \right), \text{CLIP}(\text{EncDesc}) \right]$$

---

Equation 5: Backpropagation of the loss corresponding to one iteration in the optimisation loop.

---

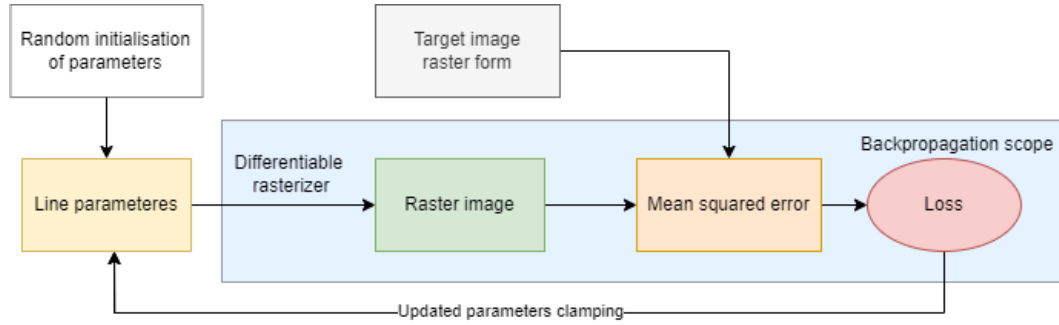
## 4.2 Implementation of the new text-to-drawing algorithm: CLIP+DiffDrawing&Sketching (CLIP+DDS)

At the beginning of the implementation, the differentiable rasteriser from [3.10](#) was experimented with to understand the algorithm better, using the provided Jupyter notebooks with PyTorch code from: <https://github.com/jonhare/DifferentiableSketching>. The notebooks explained the use of gradient descent to optimise parameters of points, lines and curves to match a target image. As the examples were simple, they could be run locally on a laptop. To fully understand the PyTorch code in Differentiable Sketching and CLIPDraw, tutorials from the module COMP6248: Deep Learning and the Official PyTorch website (<https://pytorch.org/tutorials/>) were completed.

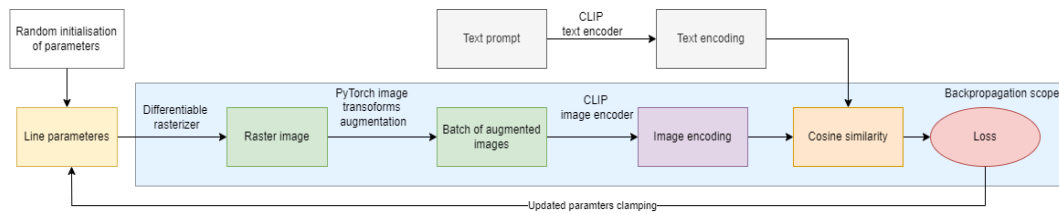
Then, the code from the Differential Sketching library *imageopt.py* was modified. As the differentiable rasteriser minimises the loss between the generated raster and the target images, the optimisation function in the file had to be adjusted to follow the structure of the *Algorithm 1*:

- CLIP model had to be loaded
- The target image was replaced by the CLIP encoding of a text prompt
- The generated raster image had to be preprocessed before being passed to CLIP as it needed to be normalised and resized to the size of 224x224

- The generated raster image had to be encoded by the CLIP image encoder before being passed to the loss function
- PyTorch image augmentation had to be added



Original optimisation algorithm



Modified optimisation algorithm

Figure 8: Diagrams of the original optimisation algorithm from [3] (top) and the updated version, corresponding to *Algorithm 1* (bottom). After the backpropagation step, all the parameters had to be clamped into the correct interval as it is possible that the line position can be forced outside of the image’s bounds.

### 4.3 Challenges faced during the implementation

The CLIPDraw and Differentiable Drawing and Sketching were initially set up to run on a laptop with an Nvidia GTX 1660Ti GPU. However, as only it only had 6GB of GPU VRAM available, not enough memory could be allocated, which stopped the execution. Therefore, a different setup had to be used.

After the guidance of a supervisor, the university GPU server: *yann.ecs.soton.ac.uk* was used. The server had GTX 1080Ti GPUs with 11GB of VRAM available. This worked well for CLIPDraw, and experimentation with the algorithm was performed on the server during the first half of the project’s timeline. However, after later completing the *CLIP+DDS*, it was discovered that the algorithm’s memory requirements are even higher.

Thus, a different approach was taken: using the Google Colaboratory (Colab) platform (<https://colab.research.google.com>) as the main compute service. With the Pro subscription, Nvidia Tesla T4 and Nvidia Tesla P100 GPUs (both equipped with 16GB of VRAM) were available. This amount of VRAM was sufficient enough for the new algorithm.

It also had significant advantages compared to the university GPU servers, as it had direct Google Drive access without the need for an API. This was essential as the generated images could be saved on Google Drive and easily accessible in other tools such as Google Sheets during experimentation and comparison stages. Additionally, up to 3 concurrent GPU runtimes were available, which was particularly useful during the comparison process as it required a high amount of generated images to gather meaningful data. Finally, it was also the simplest to set up.

Due to the platform's advantages, it was used as the primary GPU service throughout the whole project.

Unfortunately, the GPU resources for Google Colab were not guaranteed (there was a possibility that sometimes no GPUs would be available). Therefore, the Gradient platform (<https://gradient.run/>) was used as a backup service as it also provided GPUs with 16GB of VRAM.

As both services utilised Jupyter notebooks, the code for the new algorithm had to be modified accordingly.

The cost of the Colab Pro subscription was ~£9.72 per month, and this tool was used for four months. This expense was definitely worth the investment, as it provided many advantages which significantly improved the process of gathering comparison data and experimentation.

## 5 Experimentation

Once the algorithm implementation was finished, the experimentation and testing phase began. This phase aimed to check if the algorithm works well and explore possible improvements for the quality of images.

The first runs of the algorithm were performed without any colour or width optimisation, fixing bugs such as mismatches between tensor shapes. Afterwards, modifications were made so that each stroke’s width and colour could be optimised independently.

As explained in [3], the *over* composition function had to be used during rasterisation when using colour. It respects the order of the images during the composition strokes and therefore accounts for strokes that go over others.

During the experimentation, it was observed that with the resources available, only approximately 220 curves with two control points could be generated. Although curves are more flexible than lines, the generated images looked empty as many whitespaces were left. The algorithm primarily focused on the object’s outline when using curves, as it does not have enough curves to work with. Therefore, the image generation was constrained to lines only as images could be reliably generated without memory issues when using 850 lines. This constraint was adopted throughout the entire project.



---

Figure 9: The difference between the images generated using the maximum available memory with the text description: *A watercolour painting of a cat*. The image on the left was generated using 220 curves, while 850 lines were used to generate the image on the right.

---

The width of each line was controlled with the  $\sigma^2$  parameter of the differentiable relaxation of a line (*Equation 2*). Therefore, it is possible to learn each width of a line separately by storing the  $\sigma^2$  parameters in a tensor, passing it to the rasterisation function during the rasterisation process.

### 5.1 Hyperparameters tuning

The algorithm had many parameters that could be tuned to improve the quality of the images. However, this also meant that the parameter space was large, and some hyperparameters which did not have a significant effect were explored less due to the time constraints.

### 5.1.1 Methodology

To find the best parameters, an objective measurement of the image’s quality had to be made. Initially, the hyperparameters were tuned based on a visual inspection. Once rough values of the parameters were obtained, an objective approach had to be chosen to minimise the human bias.

Thus, the CLIP model was used to score the similarity between the generated image and the output. Similarly to the loss function, the cosine similarity between the text encoding and the generated image encoding was calculated. The advantage of cosine similarity is that the outputs are in the interval  $\langle -1, 1 \rangle$ , so it was easy to determine which images had better quality.

The following hyperparameters were tuned:

- Learning rate of the positional parameters (start, end parameters of a line)
- Learning rate of the RGB colour parameters of each line
- Learning rate of the  $\sigma^2$  parameter
- Initial value of the  $\sigma^2$  parameters of each line
- Number of iterations
- Number of lines

In the beginning, all lines were initialised to have a maximum width and the widths were modified throughout the optimisation process. This approach was chosen to force the algorithm to fill the images with “paint” instead of focusing primarily on the object’s shape.

All learning rates depended on the number of iterations, as with the more iterations, lower learning rates could be chosen to achieve a better optimum. 1000 iterations were selected due to time constraints as they provided both decent results and took ~23 minutes on the most frequently assigned Nvidia Tesla P100 GPU. 850 lines were chosen as it was the highest reliable value without any memory issues since more lines produced better looking, “fuller” images.

|  |       |
|--|-------|
| Learning rate of the positional parameters | 0.004 |
| Learning rate of the RGB colour parameters | 0,005 |
| Learning rate of the $\sigma^2$ parameter  | 0,001 |
| Initial value of the $\sigma^2$ parameters | 15.0  |
| Number of iterations                       | 1000  |
| Number of lines                            | 850   |

---

Table 1: Hyperparameters for the best image quality discovered during the tuning process.

---

## 5.2 Possible improvements to the algorithm

Several techniques aiming to improve the quality of generated images were experimented with. The same approach (using the CLIP score) as in the previous section was taken to determine the effectiveness of the techniques.

### 5.2.1 Image augmentation

Similarly to the behaviour observed in [1], when no augmentations were used throughout the image optimisation process, the results achieved good numerical values; however, to the human eye, the images looked chaotic and did not resemble any real-life objects. This effect could be described as overfitting.

As explained in [36], data augmentation is a popular technique that is used to *enhance the size and quality of training datasets such that better DeepLearning models can be built using them*. The purpose of data augmentation is to provide a more comprehensive training set and minimise overfitting issues. Although the drawing algorithm does not require a training set, using multiple augmented images in the optimisation process helps create pictures with a clearer structure.



---

Figure 10: The effect of overfitting: The image description was: *A watercolour painting of a cat*. The image on the left was generated with no augmentation, while the same augmentation technique described in [1] was used when generating the image on the right. Both images received similar CLIP scores,  $\sim 0.3$  (left) and  $\sim 0.4$  (right); however, the structure on the right is much better as it resembles a cat.

---

Since the authors of CLIPDraw left the effects of augmentations for further research [1], the transformation techniques from the *torchvision* library presented in [Appendix C](#) were used to observe how different augmentations impact the image’s composition.

#### 5.2.1.1 Evaluation

Techniques which only affect the image’s colour (*ColorJitter*) or sharpness (*GaussianBlur*) did not impact the overfitting problem as they achieved high scores with fuzzy images. On the other hand, augmentations that modified the image’s composition, such as *Crop*, *Rotation*, *Perspective*, or *AffineTransform*, successfully minimised the overfitting.

If overused, they present a different problem: artefact creation (multiple heads, eyes, or an upside-down position). Therefore, there is a tradeoff between the overfitting limitation and artefact creation, which has to be controlled by modifying the probabilities of the random augmentations.

Overall, the composition of *RandomPerspective* and *RandomHorizontalFlip* + *RandomPerspective* seemed to have achieved the best visual results, and the image



structures were comparable with the structures of the images created using augmentations used in CLIPDraw.

### 5.2.2 Grey-colour initialisation

In CLIPDraw, a value for each RGB channel was set randomly during the initialisation step. In theory, the randomness effect might cause a scenario where the colour of a line would have to be changed to the opposite value. The algorithm might then take longer to achieve the desired colour value.

A different approach was attempted: all channels were initialised to the value in the middle of the available spectrum: (0.5, 0.5, 0.5) in the interval  $\langle 0, 1 \rangle$ , corresponding to a grey colour. As the value is exactly in the middle of the colour spectrum's interval, the correct values can be reached faster throughout the optimisation. To test whether this modification improves numerical quality, 21 images were generated using the following combination of styles and objects:

Styles:

- “a watercolour painting of”
- “an impressionist painting of”
- “a drawing of”
- “a cubist painting of”
- “a realistic photograph of”
- “a 3D rendering of”
- None

Objects:

- “a cat”
- “a flower”
- “the Eiffel Tower”

#### 5.2.2.1 Evaluation

The average loss using randomly generated colours was  $\sim -1.80$ , and the average cosine CLIP similarity of the encoding of a final image with an encoding of the original text prompt was  $\sim 0.43$ . If all lines were initialised with a grey colour, the average loss was slightly lower (better) at  $\sim -1.84$ , and the cosine CLIP similarity of the corresponding image-text encoding pair was marginally better at  $\sim 0.44$ .

Although there was a visible difference in the images' colour scheme, the numerical difference between the average CLIP similarities of the two initialisation strategies was lower than  $0.01$ . Therefore, it can be concluded that this improvement does not impact the quality of the images.

The minor difference also strongly depends on the learning rate used for the lines' colour, as with a higher learning rate, random initialisation would match the grey initialisation. If left running for enough iterations, both techniques should produce images with similar

colour schemes. The randomness of the initialisation of coordinate positions could also slightly impact the measurements (Section 5.3).




---

Figure 11: The visual difference between the images generated with random initial colours (left) and uniform grey colour (right). The text description was: *a cubist painting of a cat*.

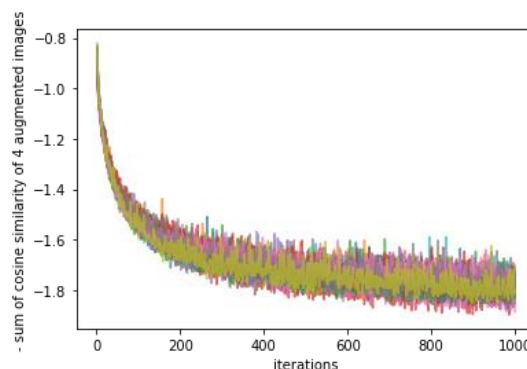
---

### 5.3 The impact of a random seed value on the quality of the generated images

Although the randomness of the colour during the initialisation can be eliminated using the “middle” RGB channel values, the heuristics for line positions are more complicated as there are multiple factors that need to be taken into account: the number of lines used, their length and position. Therefore, the line parameters were left to be generated randomly, using the `make_init_params()` function in the `imageopt.py`[3] file.

Moreover, since the majority of the image augmentation functions available in the `torchvision.transforms` library are based on the properties of randomness; the algorithm’s random aspect would be challenging to avoid.

To observe how the image quality and the loss change with a random seed, 19 images with the text description: *a watercolour painting of a landscape*, using 1000 iterations and 850 lines with random seeds in the range of 1-19 were generated.

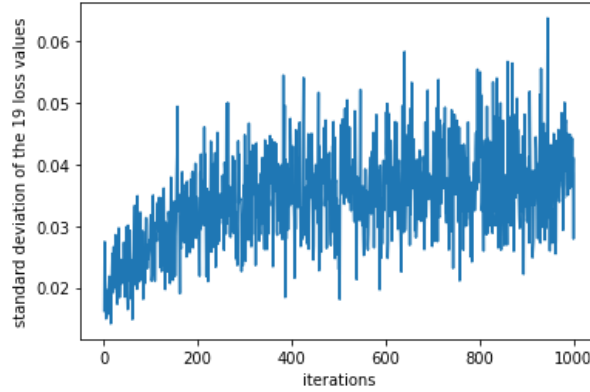


---

Figure 12: A sample of 4 of the generated images (top) and the graph of the loss of the 19 runs with different seeds (bottom). Although the structure of the images varies substantially, the loss values follow the same pattern.

---

As shown in *Figure 12*, the random seed value significantly affects the image's visual composition, but the loss stays relatively the same. The standard deviation of the loss values was calculated between all 19 images at every iteration step to further inspect how the seeds affect the loss.



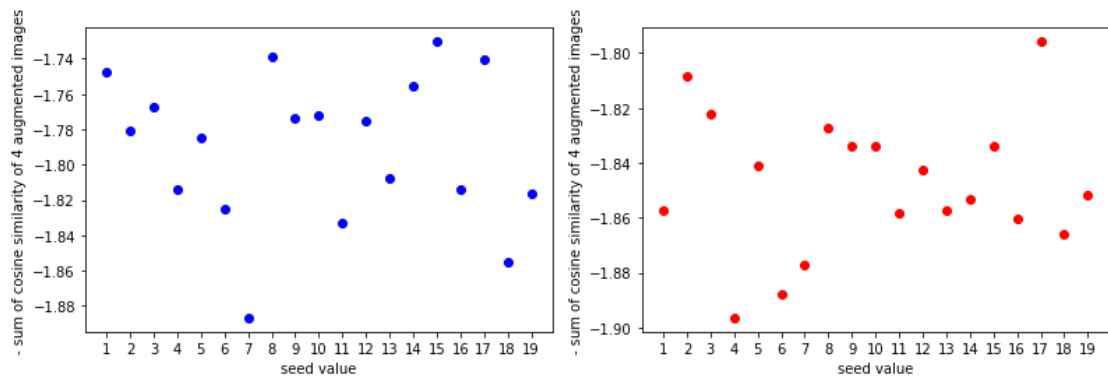

---

Figure 13: The graph of the standard deviation between 19 loss values corresponding to the generated images per iteration.

---

### 5.3.1.1 Evaluation

As illustrated in *Figure 13*, the loss variation between the 19 images is insignificant as the highest value is only approximately 0.06. It is interesting to observe that the standard deviation of the loss slightly increases in the later stages of the iterations. The average standard deviation between the generated 19 images is  $\sim 0.0344$ . Again, this is relatively insignificant, as the average loss was  $\sim -1.79$ , and it can be concluded that the random initialisation does not affect the numerical quality of images. The variation can be decreased further by using the image with the lowest loss in the last 100 iterations, reducing the standard deviation to  $\sim 0.0247$ .




---

Figure 14: The difference in the range of the loss values of the final image (left) and the image with the lowest loss in the last 100 iterations (right). As the range of the y-axis of graphs suggests, saving the image in the final 100 iterations slightly decreases the range of the loss values.

---

## 6 Comparison with CLIPDraw

In parallel with the experimentation process, a thorough comparison between the new algorithm (CLIP+DDS) and CLIPDraw was performed to find how the performance between these algorithms varies. Grey-colour initialisation ([5.2.2](#)) was chosen for the new algorithm as it achieved more realistic colours without affecting the numerical quality.

### 6.1 Methodology

Throughout the comparison process, Jupyter notebooks running on the Google Colab service were used to run these two algorithms. To obtain the best results for a fair comparison, the following constraints were made:

- algorithms used 425 and 850 lines and performed 1000 steps (iterations) of the optimisation loop
- the augmentation techniques used were `torch.transforms.RandomPerspective` and `torch.transforms.RandomResizedCrop`
- no negative prompts[1] were used

In addition to the resource-limited 850 lines, 425 lines were chosen to determine how the algorithms compare when lower resources are available.

The generated images were saved on Google Drive and displayed in a Google Spreadsheets document along with their perspective losses and additional information such as time taken, hyperparameters, and text description used. A service bot account was used to record the data.

The following six image styles were chosen:

- “a watercolour painting of”
- “an impressionist painting of”
- “a drawing of”
- “a cubist painting of”
- “a realistic photograph of”
- “a 3D rendering of”
- None

The image styles were chosen based on the observations in [1] and art techniques that could work well when using lines during image generation.

These prompts were then paired with the following image descriptions:

- “a cat”
- “a fantasy world”
- “a castle”
- “a landscape”
- “a city”

- “a beautiful flower”
- “a formula 1 car”
- “a pirate ship”
- “a medieval empire”
- “a modern society”
- “happiness”
- “love”

A variety of both material and abstract descriptions were chosen to observe if one algorithm works better with specific groups of prompts than the other. (Samples of the generated images can be found in [Appendix F](#).)

### 6.1.1 “Calibration” of the algorithms’ hyperparameters

The best effort was made to give the algorithms equal conditions. This was achieved by “calibrating” the hyperparameters so that the shape of their loss graphs became similar. This approach also removed any possible advantages caused by the different initialisation techniques, as the colour learning rate parameters were modified too. Furthermore, a CLIP encoded reference image was used instead of text prompts, which was useful when visually observing how the hyperparameter tuning impacts the generated image with respect to the reference image. Once the loss graphs had similar shapes (*Figure 15*), the hyperparameters shown in *Table 2* were selected.

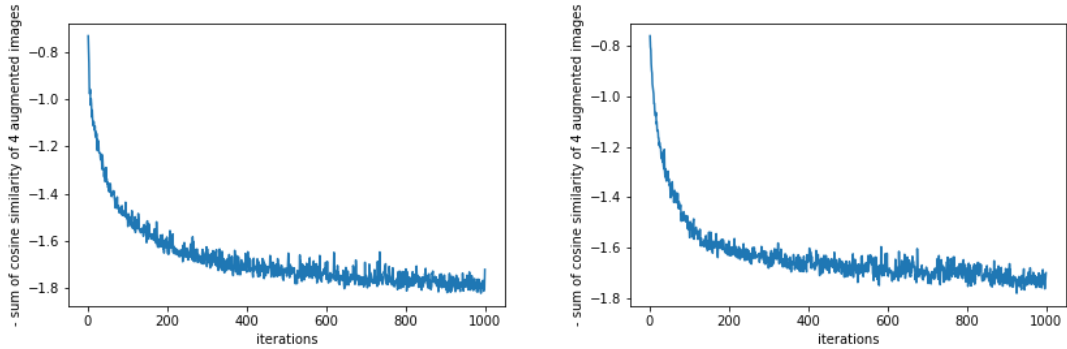


Figure 15: Example of loss graphs of CLIPDraw (left) and the implemented algorithm (right) after calibration.

|  | CLIPDraw | CLIP+DDS |
|--|----------|----------|
| Learning rate of the positional parameters | 0.73     | 0.002    |
| Learning rate of the RGB colour parameters | 0.007    | 0,003    |
| Learning rate of the width of the lines    | 0.07     | 0,001    |
| Initial value of the $\sigma^2$ parameters | N/A      | 15.0     |

Table 2: The hyperparameters chosen during the calibration process.

## 6.2 Evaluation of the results

To offset minor differences that could be affected by a random seed, two images were generated for both 425 and 850 iterations per each *style+prompt* combination. In total, 336 images were created with each algorithm.

### 6.2.1 Resource efficiency

As illustrated in *Figure 16*, CLIPDraw is more efficient since CLIP+DDS required more time and memory. On average, ~273% more time and ~123% more VRAM<sup>1</sup> were required when running on 1000 iterations using 425 lines.

With the increased number of lines, CLIP+DDS scales much worse compared to CLIPDraw in terms of time taken, as it took 535% longer. Furthermore, the memory requirements of the new algorithm were ~211% higher<sup>1</sup>.

This efficiency gap is mainly caused by the differences in the performance of the rasterisers used by the algorithms.

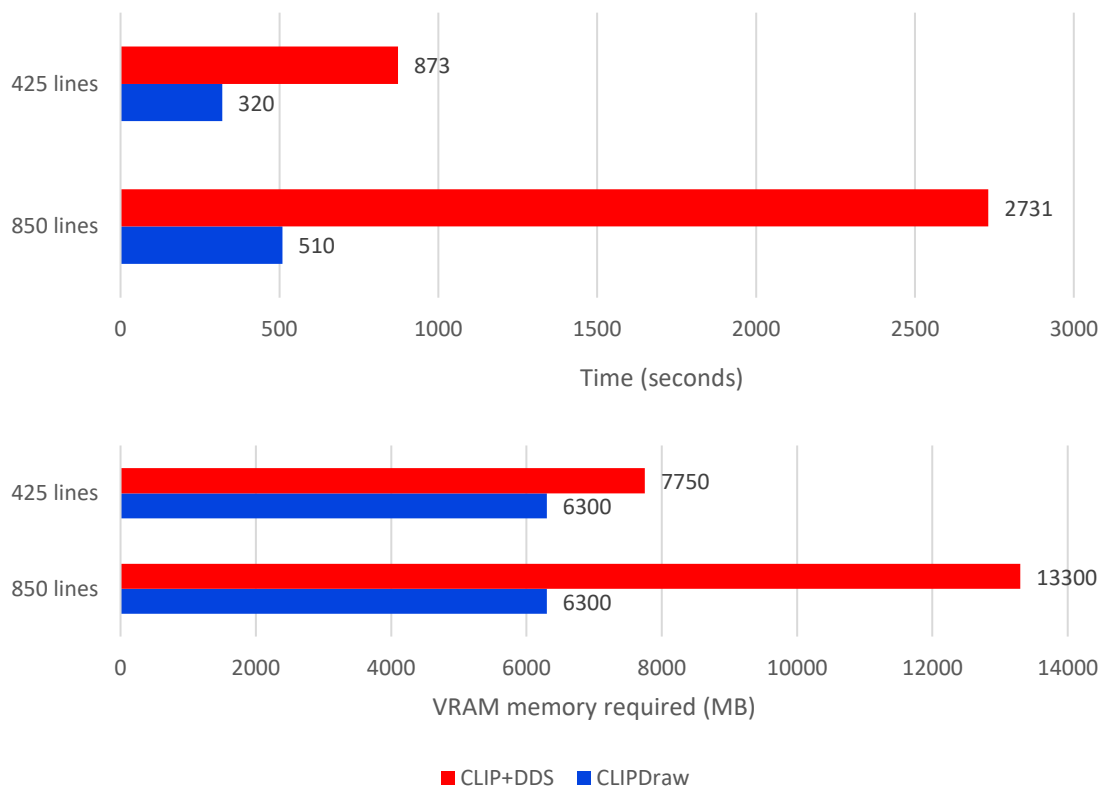


Figure 16: The comparison of time (top) and VRAM (bottom) needed for image generation<sup>1</sup>. (Memory allocated by the CLIP model is also included)

<sup>1</sup> Nvidia Quatro P5000 GPU with 16GB VRAM was used to run both algorithms.

## 6.2.2 Image quality

To objectively measure image quality, the same approach as in section 5 was used as the quality metric. As the loss fluctuates throughout the optimisation process, the image with the lowest loss in the last 100 iterations was selected instead of the image generated in the last iteration.

The average quality score of the 336 images generated by CLIPDraw was  $\sim 0.400$ . CLIP+DDS achieved a slightly higher average score at  $\sim 0.428$ . Thus, overall, the new algorithm was around 7% better at generating images that match the given text description.

The performance did not significantly differ with respect to the object specified in the description prompts as per all but one description; the error stayed within a  $\pm 1\%$  margin. The only outlier was the prompt “love”, where the performance of the algorithms got very close, as only a  $\sim 2.7\%$  difference was observed.

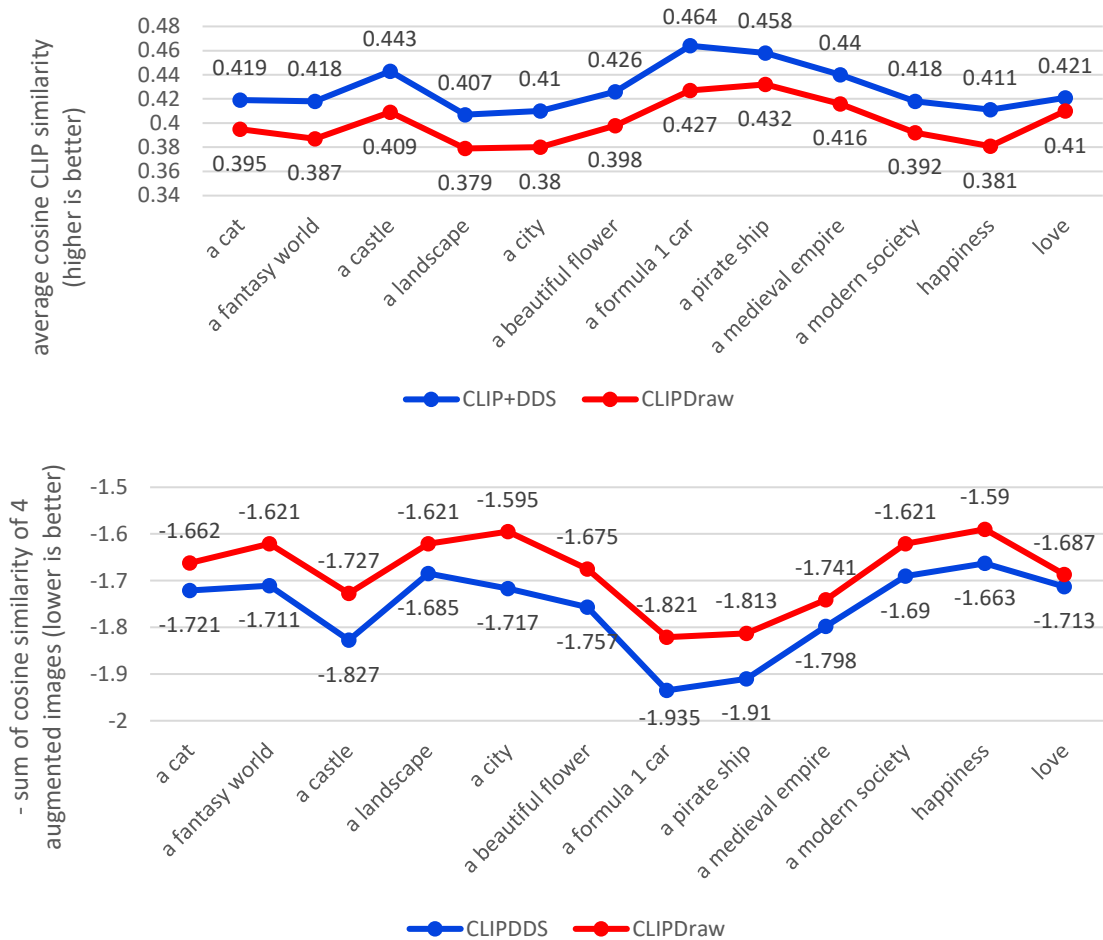


Figure 17: The difference in average image quality (top) and the average losses achieved during the generation process (bottom) based on the **specified objects**. It is not surprising to observe that the rate of difference between the average loss and the image quality per group of the two algorithms is almost identical since both use the cosine similarity of the CLIP scores.

Similar results could be observed when fixing the image styles. The variance of the differences between the two algorithms was again within the  $\pm 1\%$  margin. The lowest dissimilarity was found when no image style was used, at  $\sim 4\%$ .

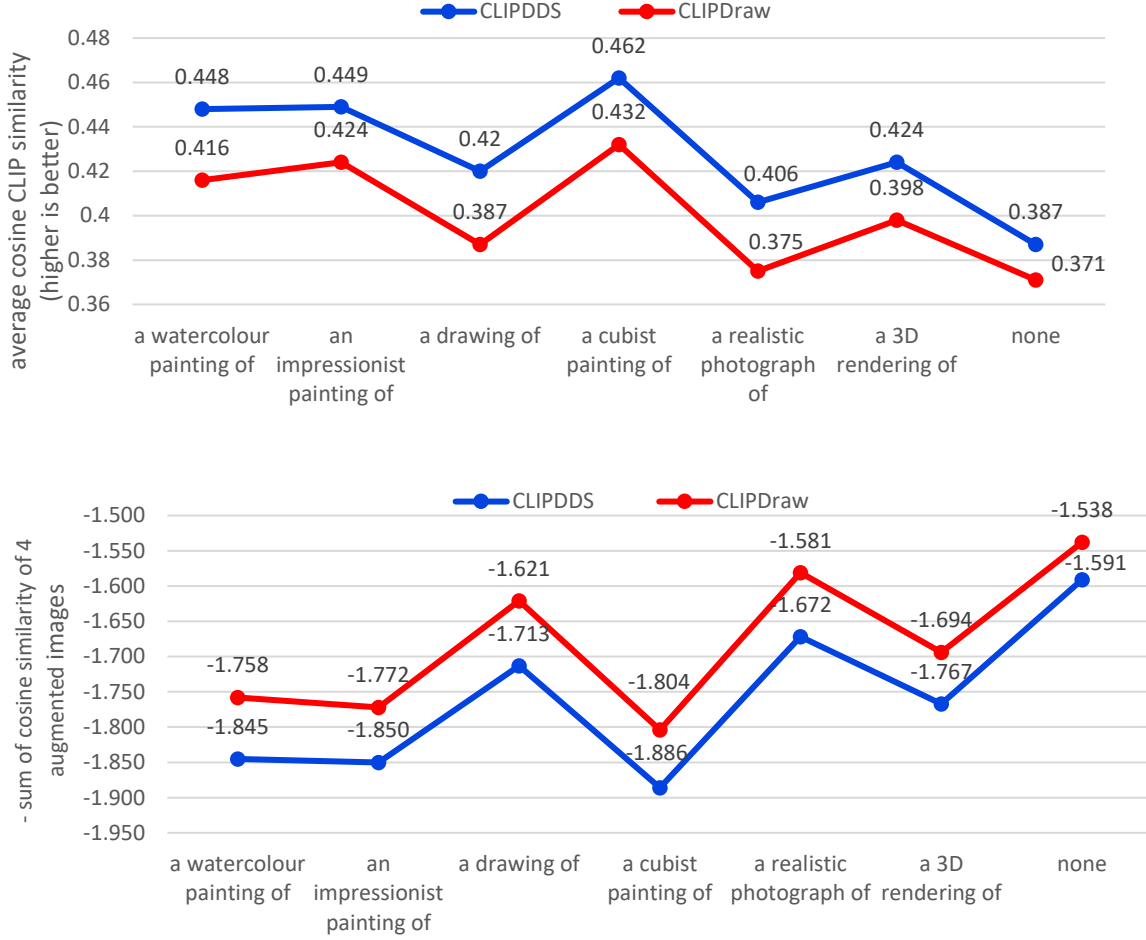


Figure 18: The difference between the average image quality (top) and the average losses achieved during the generation process (bottom) based on the **specified image styles**. The relation between the average loss and the image quality is apparent.

Both algorithms achieved better results with the higher number of lines, although this difference was minor: at  $\sim 2\%$  improvement for CLIPDraw and  $\sim 3.5\%$  improvement for CLIP+DDS. Similarly, there was a slight difference between the two algorithms w.r.t. the number of lines used. The new algorithm outperformed CLIPDraw by  $\sim 6.1\%$  on 425 lines and  $\sim 7.6\%$  on 850 lines.

Therefore, if there are enough resources, the quality of images generated by CLIP+DDS scales better with a higher number of lines.

|                  | CLIPDraw | CLIP+DDS |
|------------------|----------|----------|
| <b>425 lines</b> | 0.396    | 0.421    |
| <b>850 lines</b> | 0.404    | 0.435    |

Table 3: The average cosine CLIP similarity of the algorithms w.r.t. the number of lines used.



### 6.2.3 Discussion of the results

The algorithms are very close in terms of the metric chosen in [6.2.2](#). It should be noted that even though the calibration process took place, it is challenging to determine the equality of the hyperparameters for an entirely fair comparison since both algorithms use different rasterisers with individual hyperparameter values.

However, as the new algorithm's performance slightly but consistently outperformed CLIPDraw, it can be said with certainty that the algorithms are *at least equal in the quality of images produced if resources are not considered*.

On the other hand, CLIPDraw was notably more efficient in time and memory resources. For this reason, it could outperform the new algorithm if the constraints (425 and 850 lines) were lifted and the only restriction taken into account were resources available.

### 6.2.4 Challenges faced

When comparing the algorithms, the major challenge was the time required to obtain enough data to produce a conclusion with high confidence. Approximately 174 hours of constant runtime on the Google Colab Pro with a single NVIDIA P100 GPU was needed for CLIP+DDS. However, as the tool was mainly designed for interactive use, only 12 hours of runtime were usually available each day. Thus, when using three separate instances, the data-gathering process for the algorithm took approximately a week.

Furthermore, due to updates made in the Google Colab virtual machines, after the first day of runtime it was not possible to run CLIPDraw on the Colab platform as the rasteriser installation was failing.

Instead, a backup service: Gradient, was used. As it did not have direct access to Google Drive, modifications to the source code had to be made, so the generated images were still visible in Google Sheets. The service also had less powerful GPUs (Nvidia Quadro P5000), but due to the better efficiency of CLIPDraw, this was not a significant issue.

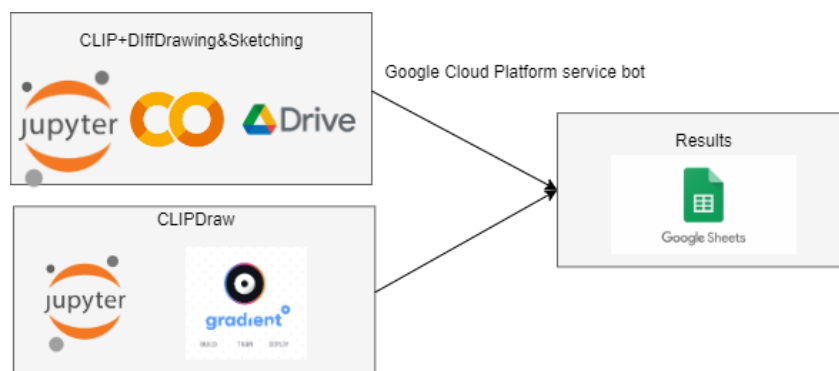


Figure 19: The diagram of services used throughout the testing process. Images were taken from: <https://commons.wikimedia.org/> and <https://github.com/Paperspace/gradient-cli>

## 7 Practical application of the algorithm

It was shown that the new algorithm could produce images with good quality. This section will highlight one of the possible applications of text-to-image synthesis algorithms. As it requires text prompts to generate an image, a domain with textual information was needed.

An obvious domain was fiction literature since it can contain images (illustrations) related to the paragraph's current story or chapter. Because the image generation algorithm only requires a few words as an input, an NLP pipeline that would create headings from large pieces of text from books had to be developed. This pipeline was then integrated into a web application that abstracted the algorithms from the user.

### 7.1 A high-level overview of the implementation

The following pipeline was created to capture the text's main points in a short heading. First, a chapter summary with a maximum of 450 words is generated using a Longformer Encoder-Decoder model[37]. The summary is then fed into two statistical and two machine learning models that create four different image descriptions. The most-suited image description can be chosen and used as an input to generate a drawing.

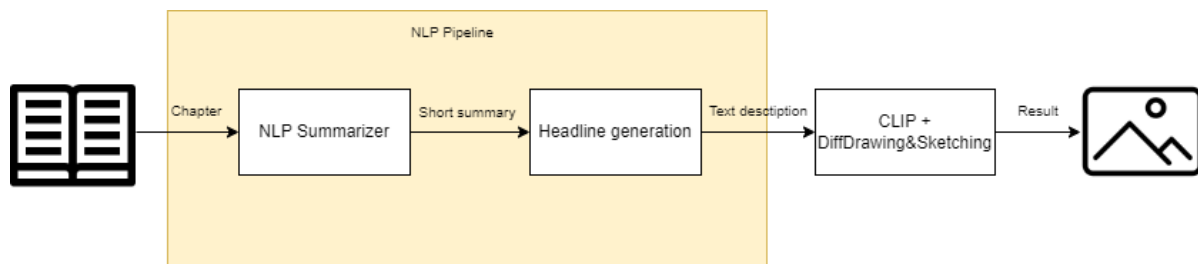


Figure 20: A diagram of the designed system to generate illustrations from the text.

### 7.2 NLP models used

Various SOTA NLP models were considered when researching a model suitable for the summarisation and headline generation process. The database in the *huggingface* library (<https://huggingface.co/>) was used as it provided a variety of models suitable for the pipeline.

#### 7.2.1 Summarisation

Since the length of book chapters can span multiple pages, a model that could use such a long piece of text as input had to be used. The Longformer Encoder-Decoder (LED)[37] was developed to improve the scalability of the attention mechanism used in the Transformer-based models and enable the processing of long documents.

Compared to other popular models that usually only take around 512-1024 tokens as an input, the LED-large model can work with 16 384 tokens while achieving SOTA results on certain datasets.

A fine-tuned version of this model (<https://huggingface.co/pszemraj/led-large-book-summary>) on the BookSum dataset[38] was used within the pipeline. The BookSum dataset contains book paragraphs and chapters from the Project Gutenberg book repository (<https://www.gutenberg.org/>) and human-written summaries on a paragraph, chapter, and book level. As the main domain of the inputs for the summariser will be literature, the fine-tuned model was the option with the best potential performance.

## 7.2.2 Headline/Keyword extraction

Multiple models were used to generate a text description from the summary to provide options to the user. Two unsupervised statistics-based algorithms for keyword extraction were used:

- **YAKE**[39], which relies on *statistical text features of the text*[40]
- **RAKE**[41], based on the observation *that keywords frequently contain multiple words with standard punctuation or stop words*[42].

Two machine learning-based algorithms were also utilised:

- **T5-base-en-generate-headline** (<https://huggingface.co/Michau/t5-base-en-generate-headline>): a fine-tuned version of Google’s T5 general-use NLP transformer model[43] on 500 000 articles with headings.
- **KeyBERT** (<https://github.com/MaartenGr/KeyBERT>): a keyword extraction algorithm that uses BERT[44] embeddings and cosine similarity to find similar words or phrases to the document.

To find whether there exists a correlation between the algorithm used and the quality of the generated image, ~220 summarisations from the BookSum dataset were generated, and each algorithm created its version of the image description. The generated images and text descriptions were then scored by CLIP. The keyword extraction algorithms (YAKE, RAKE, KeyBERT) were limited to 8 keywords, while T5-headline was left at the default limit as it sometimes tended to produce full questions or sentences.<sup>2</sup>

| <b>T5-headline</b> | <b>YAKE</b> | <b>RAKE</b> | <b>KeyBERT</b> |
|--------------------|-------------|-------------|----------------|
| 0,439              | 0,407       | 0,428       | 0,435          |

---

Table 4: The CLIP scores of roughly 220 images and text descriptions generated by the corresponding algorithms. The statistics-based methods, especially YAKE, seem to perform slightly worse than machine-learning-based ones. T5-headline had a slight advantage as it sometimes used more than eight words.

---



---

<sup>2</sup> The link to the database of all generated text and image results is in the section [12.6.1](#)

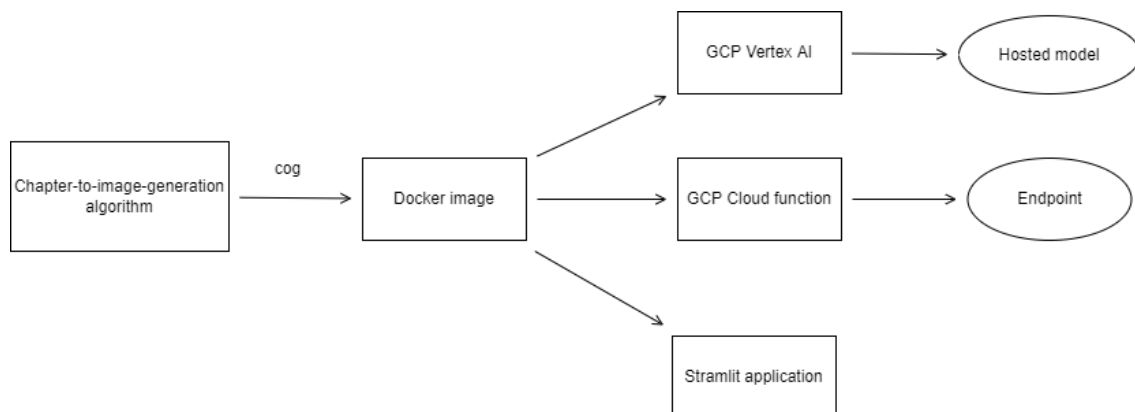
## 7.3 User interface design and implementation

To abstract the complicated system from the user, a simple UI was designed and later developed using the *streamlit* library (<https://streamlit.io/>) ([Appendix E](#)). Wireframes were used for the design process ([Appendix D](#)).

Users can select the number of lines, iterations and whether a text description or a book chapter will be used. Based on their selection, either the NLP pipeline will be utilised and the generated headlines from the different algorithms will be shown, or the generation process will start immediately. A new screen for each step was designed to guide the user through the process to make these steps straightforward. Once the generation is done, the image, graph of the losses and a video made of iteration snapshots are displayed.

## 7.4 Deployment

Initially, the finished pipeline was planned to be uploaded to the Google Cloud Platform's GPU virtual machines in the form of a docker image created using the *cog* library (<https://github.com/replicate/cog>). *Cog* makes it simple to build docker images for machine-learning algorithms. HTTP requests from the website could then be made to the deployed image to utilise the pipeline (*Figure 21*).



---

Figure 21: The diagram of the initial solution

---

Unfortunately, the model deployment process is constrained to the typical neural network models, where the weights can be saved to a file. Only experimental approaches are currently available for custom online predictions, and the learning curve for those was too high given the available time. This problem persisted with other cloud platforms such as Amazon AWS and Microsoft Azure.

Therefore, a different approach was taken: the two Jupyter notebooks with the drawing algorithm and the NLP pipeline were kept separate (as the NLP pipeline required GPU VRAM), and communication between them was made using JSON files.

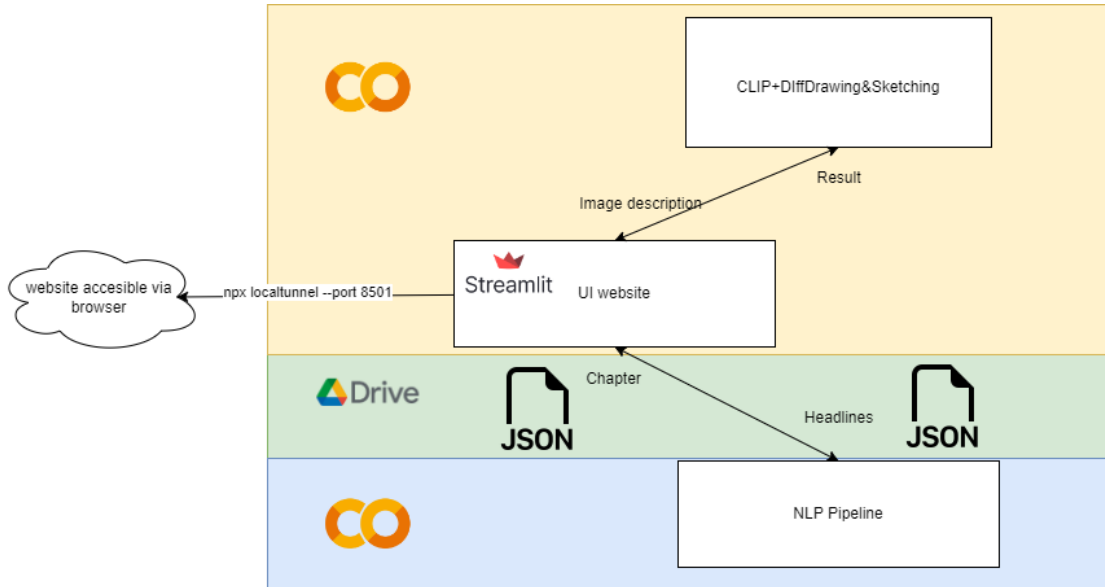


Figure 22: The diagram of the final solution. Images were taken from: <https://commons.wikimedia.org/> and <https://streamlit.io/brand>

As shown in *Figure 22*, the website code and the drawing algorithm are set up on one Colab instance, while the NLP pipeline is hosted on a second instance. The JSON files used for communication are saved on Google Drive. As the website is only available on a local network of the Colab virtual instances, the *localtunnel* package (<https://www.npmjs.com/package/localtunnel>) was used to expose the localhost website to the internet. This system was made with the aim to be used by a single user only, as not enough GPU resources are available for concurrent runs.

## 7.5 Testing

Finally, the complete solution was thoroughly tested to find bugs in any part of the process. Manual testing was used for all tests. The testing of UI elements was performed first to spot errors and bugs of the website's front-end. Once the website testing was finished, more manual tests were performed on the whole pipeline using the UI.

## 7.6 Drawing completion

Another application: the completion of drawings using text prompts, was also explored. Initially, the DDS rasteriser was used to create a vectorised version of the given raster drawing. The position parameters of the lines were stored after the final iteration. During the initialisation process of CLIP+DDS, the parameters were exported, and a given number of randomly initialised lines was added. The drawing was then optimised to match the text description.

There was a major issue with this approach (*Figure 23*), as the structure of the original drawing was lost during the optimisation process. The solution was to “freeze” the parameters of the initial picture. This worked better, as shown in *Figure 24*; however,

the quality of images varied, as it is difficult to specify positions or fine details. This is possibly caused by the problems of the CLIP model mentioned in [3.8](#).



---

Figure 23: The initial drawing (left) (taken from [3]), vectorised version with 500 lines using DiffDrawing&Sketching (middle) and a result of the description: *Homer Simpson in a Desert* using additional 350 lines.

---



---

Figure 24: (left) The result of the same process shown in *Figure 23* performed with “frozen” parameters. (right) An example of a drawing with positional problems, the description used was: *Homer Simpson and Peter Griffin*.

---

## 8 Critical evaluation

The goals that were set for the project were achieved. The implementation of CLIP+DDS performed very well as it had better results than CLIPDraw. Although the deployment of the final pipeline of the algorithm and the NLP models had to be modified, a decent workaround has been developed.

Unfortunately, no techniques that would further improve the quality of the generated images were found in the experiments performed in [5.2](#), and more in-depth research would need to be conducted.

It can be argued that the quality of the generated images is quite blurry: this is a characteristic of the rasteriser algorithm discussed in [3]. While images in the vector form do not have this problem, they contain many whitespaces. When the vector lines are widened to remove the whitespaces ([Appendix G](#)), a loss of detail occurs. Once the resources allow a more significant number of lines to be used, a wider area could be covered, and the blurriness would decrease. Furthermore, the vector images become “fuller” and more detailed. Thus, more resources or better optimisation of the rasteriser are needed for the vector images to be considered as a replacement for the raster images.

Based on the experience obtained, a few things would be done differently:

- More time would be invested into the setup of the development tools as they had to be changed and the codebase had to be migrated to a new platform, slowing down the progress.
- The implementation phase would take a higher priority at the beginning of the project, leaving more time for the comparison, experimentation and application phases as the algorithms require long runtimes.
- It would be better if the experimentation and comparison phase were not done since both required the same GPU resources. Instead, the UI development could be done in parallel with the comparison phase, leaving more time to research custom model deployment on the cloud.

## 9 Conclusion and future work

This project successfully developed a new text-to-drawing synthesis algorithm that exceeded the performance of CLIPDraw[1] by approximately 7%, based on the comparison of the 336 images generated by each algorithm, using 425 and 850 lines. Multiple different experiments were performed with the new algorithm; however, no improvement that would outperform hyperparameter optimisation was found. In addition, a chapter-to-headline NLP pipeline was created using pre-trained SOTA models together with multiple different headline generation techniques, and the significance of these techniques on the generated image quality was measured. Finally, a website with a simple UI that allows easy-to-use pipeline control was developed to abstract the generation process from the user.

Given more time, further experiments with the algorithms could be performed, such as using different optimisation techniques for gradient descent, which was left out due to time constraints. Furthermore, a better deployment strategy could be researched so the website would be concurrently accessible by multiple users. It would be interesting to see how this algorithm compares to CLIPDraw if the DDS rasteriser becomes more optimised, allowing a higher number of lines or curves to be used with the same GPU resources. This could also reveal whether the quality of vector images becomes good enough to replace the raster ones.



## 10 Personal reflection

Image generation and machine learning techniques such as backpropagation were new concepts to the author at the beginning of the project. As a result, the learning curve has been quite steep but manageable, and a lot of new knowledge in this field has been obtained. Moreover, as cloud computing platforms were used throughout the project, many practical skills transferable to any computing-related project were picked up.

### 10.1 Project management

#### 10.1.1 Time management

The time management of this project has evolved. Initially, only GANTT charts were used. Later in the project, with more detailed technical tasks, a KANBAN board was chosen to be used instead.

The GANTT charts in [Appendix A](#) show the difference between the estimated and actual progress of the project. The focus of the first part of the project was background reading and understanding the PyTorch code of the algorithms. In the second part (after the semester one exam period), the implementation, comparison, experimentation and final application development processes took part. Many tasks were performed in parallel since the algorithm required a lot of time to generate the data.

#### 10.1.2 Risk management

An initial assessment of risks that could slow down the report's progress was performed before the beginning of the project. With the utilisation of Colab in the implementation stage, additional potential risk, the unavailability of the service was added as a risk and a backup service was researched ([Appendix B](#)). This had proven to be useful when Colab had issues with running CLIPDraw.

## 11 Bibliography

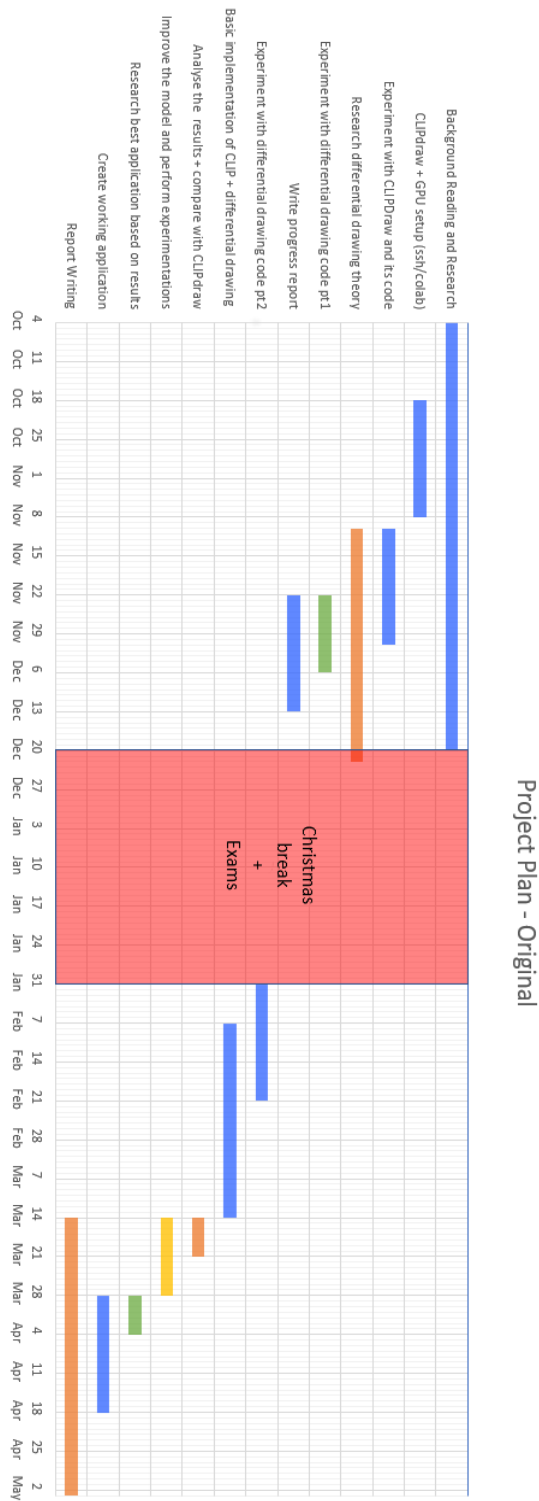
- [1] K. Frans, L. B. Soros, and O. Witkowski, "CLIPDraw: Exploring Text-to-Drawing Synthesis through Language-Image Encoders," *CoRR*, vol. abs/2106.14843, 2021, [Online]. Available: <https://arxiv.org/abs/2106.14843>
- [2] A. Radford *et al.*, "CLIP: Learning Transferable Visual Models From Natural Language Supervision," 2019.
- [3] D. Mihai and J. S. Hare, "Differentiable Drawing and Sketching," *CoRR*, vol. abs/2103.16194, 2021, [Online]. Available: <https://arxiv.org/abs/2103.16194>
- [4] M. Nixon and A. S. Aguado, *Feature Extraction and Image Processing for Computer Vision*. San Diego, UNITED KINGDOM: Elsevier Science & Technology, 2012. [Online]. Available: <http://ebookcentral.proquest.com/lib/soton-ebooks/detail.action?docID=998617>
- [5] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, vol. 32.
- [6] Z. Ma, G. Mei, and F. Piccialli, "Machine learning for landslides prevention: a survey," *Neural Computing and Applications*, vol. 33, no. 17. 2021. doi: 10.1007/s00521-020-05529-8.
- [7] Roman Paolucci, "What is a Variational Autoencoder?," *towards data science*, Jul. 2021. <https://towardsdatascience.com/what-is-a-variational-autoencoder-9b41bd63f65e> (accessed Nov. 24, 2021).
- [8] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.
- [9] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "DRAW: A recurrent neural network for image generation," in *32nd International Conference on Machine Learning, ICML 2015*, 2015, vol. 2.
- [10] E. Mansimov, E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Generating images from captions with attention," 2016.
- [11] T. Y. Lin *et al.*, "Microsoft COCO: Common objects in context," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, vol. 8693 LNCS, no. PART 5. doi: 10.1007/978-3-319-10602-1\_48.
- [12] I. J. Goodfellow *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, vol. 3, no. January. doi: 10.3156/jsoft.29.5\_177\_2.
- [13] A. Oussidi and A. Elhassouny, "Deep generative models: Survey," in *2018 International Conference on Intelligent Systems and Computer Vision, ISCV 2018*, 2018, vol. 2018-May. doi: 10.1109/ISACV.2018.8354080.
- [14] Google Developers, "Generative Adversarial Networks," May 24, 2019. [https://developers.google.com/machine-learning/gan/gan\\_structure](https://developers.google.com/machine-learning/gan/gan_structure) (accessed Apr. 12, 2022).
- [15] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," in *33rd International Conference on Machine Learning, ICML 2016*, 2016, vol. 3.
- [16] C. Wah, S. Branson, P. Welinder, P. Perona, and S. J. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," 2011.
- [17] H. Zhang *et al.*, "StackGAN: Text to Photo-Realistic Image Synthesis with Stacked Generative Adversarial Networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, vol. 2017-October. doi: 10.1109/ICCV.2017.629.

- [18] S. Arora, A. Risteski, and Y. Zhang, “Do GANs learn the distribution? Some Theory and Empirics,” *ICLR 2018 Conference Blind Submission*, 2018.
- [19] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” in *33rd International Conference on Machine Learning, ICML 2016*, 2016, vol. 4.
- [20] A. Vaswani *et al.*, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, 2017, vol. 2017-December.
- [21] M. Chen *et al.*, “Generative pretraining from pixels,” in *37th International Conference on Machine Learning, ICML 2020*, 2020, vol. PartF168147-3.
- [22] Mark Chen Alec Radford & Ilya Sutskever, “Image GPT,” *OpenAI Blog*, Jun. 17, 2020. <https://openai.com/blog/image-gpt/> (accessed Nov. 28, 2021).
- [23] A. Ramesh *et al.*, “Zero-Shot Text-to-Image Generation,” *CoRR*, vol. abs/2102.12092, 2021, [Online]. Available: <https://arxiv.org/abs/2102.12092>
- [24] OpenAI, “DALL·E: Creating Images from Text,” Jan. 05, 2021. <https://openai.com/blog/dall-e/> (accessed Apr. 12, 2022).
- [25] A. Elgammal, B. Liu, M. Elhoseiny, and M. Mazzone, “CAN: Creative adversarial networks generating ‘Art’ by learning about styles and deviating from style norms,” 2017.
- [26] L. Gatys, A. Ecker, and M. Bethge, “A Neural Algorithm of Artistic Style,” *Journal of Vision*, vol. 16, no. 12, 2016, doi: 10.1167/16.12.326.
- [27] A. Mordvintsev, C. Olah, and M. Tyka, “Inceptionism: Going Deeper into Neural Networks,” *Research Blog*. 2015.
- [28] R. G. Lopes, D. Ha, D. Eck, and J. Shlens, “A learned representation for scalable vector graphics,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, vol. 2019-October. doi: 10.1109/ICCV.2019.00802.
- [29] A. Carlier, M. Danelljan, A. Alahi, and R. Timofte, “DeepSVG: A hierarchical generative network for vector graphics animation,” in *Advances in Neural Information Processing Systems*, 2020, vol. 2020-December.
- [30] P. Reddy, M. Gharbi, M. Lukac, and N. J. Mitra, “Im2Vec: Synthesizing vector graphics without vector supervision,” 2021. doi: 10.1109/CVPRW53098.2021.00241.
- [31] T. M. Li, M. Lukáč, M. Gharbi, and J. Ragan-Kelley, “Differentiable vector graphics rasterization for editing and learning,” *ACM Transactions on Graphics*, vol. 39, no. 6, 2020, doi: 10.1145/3414685.3417871.
- [32] Jonathon Hare, “COMP6248: Differentiable Relaxations and Reparameterisations,” *Vision, Learning and Control University of Southampton*. Accessed: May 02, 2022. [Online]. Available: <http://comp6248.ecs.soton.ac.uk/handouts/relaxation-handouts.pdf>
- [33] U. of C. Department of Materials Science and Metallurgy, “What is a Tensor?” [https://www.doitpoms.ac.uk/tlplib/tensors/what\\_is\\_tensor.php](https://www.doitpoms.ac.uk/tlplib/tensors/what_is_tensor.php) (accessed Apr. 28, 2022).
- [34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, 1986, doi: 10.1038/323533a0.
- [35] D. P. Kingma and J. Lei Ba, “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION.”
- [36] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *Journal of Big Data*, vol. 6, no. 1, 2019, doi: 10.1186/s40537-019-0197-0.

- [37] I. Beltagy, M. E. Peters, and A. Cohan, “Longformer: The Long-Document Transformer.” [Online]. Available: <https://github.com/allenai/longformer>
- [38] W. Kryści *et al.*, “BOOKSUM: A Collection of Datasets for Long-form Narrative Summarization.” [Online]. Available: <http://github.com/>
- [39] R. Campos, V. Mangaravite, A. Pasquali, A. Jorge, C. Nunes, and A. Jatowt, “YAKE! Keyword extraction from single documents using multiple local features,” *Information Sciences*, vol. 509, 2020, doi: 10.1016/j.ins.2019.09.013.
- [40] Ishan Shrivastava, “Exploring Different Keyword Extractors — Statistical Approaches,” *GumGum Tech Blog*, May 11, 2020. <https://medium.com/gumgum-tech/exploring-different-keyword-extractors-statistical-approaches-38580770e282> (accessed Apr. 23, 2022).
- [41] S. Rose, D. Engel, N. Cramer, and W. Cowley, “Automatic Keyword Extraction from Individual Documents,” in *Text Mining: Applications and Theory*, 2010. doi: 10.1002/9780470689646.ch1.
- [42] Krati Agarwal, “RAKE: Rapid Automatic Keyword Extraction Algorithm,” *DataDrivenInvestor*, Apr. 08, 2020. <https://medium.datadriveninvestor.com/rake-rapid-automatic-keyword-extraction-algorithm-f4ec17b2886c> (accessed Apr. 23, 2022).
- [43] C. Raffel *et al.*, “Exploring the limits of transfer learning with a unified text-to-text transformer,” *Journal of Machine Learning Research*, vol. 21, 2020.
- [44] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” [Online]. Available: <https://github.com/tensorflow/tensor2tensor>

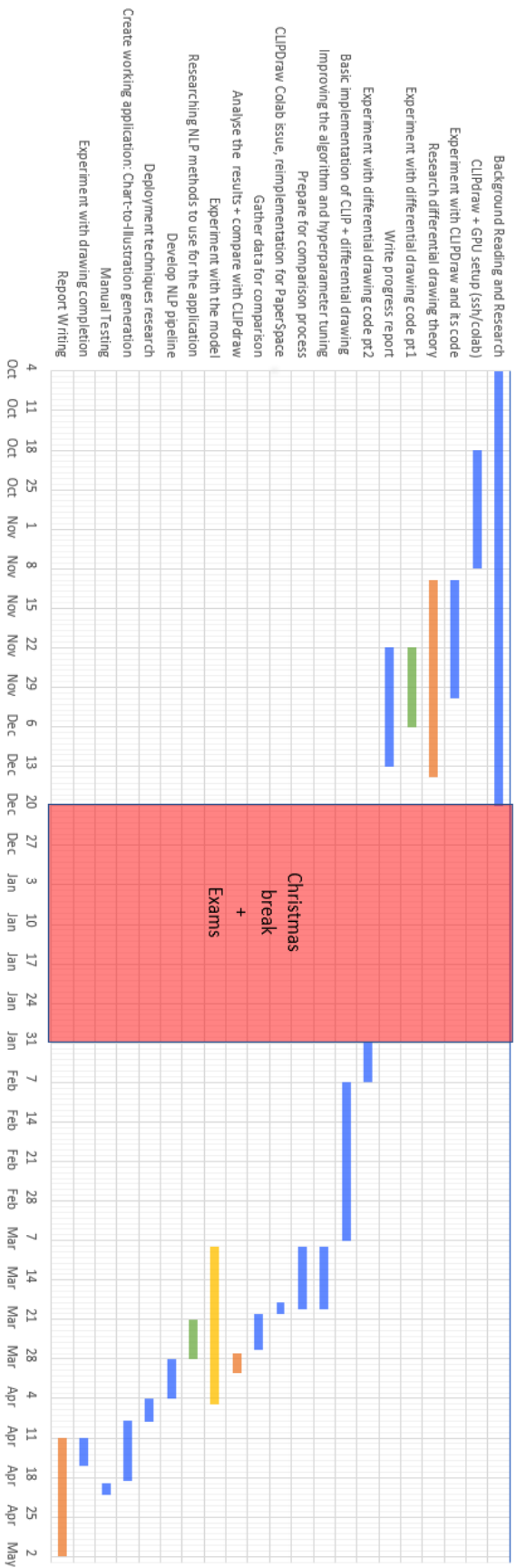
# 12 Appendix

## 12.1 Appendix A: Project Management



Appendix A1: The initial project plan proposed during the progress report.

## Project Plan - Actual



Appendix A2: The updated project plan.

To do | 5

Manual Testing

Migrate + edit background reading from the progress report  
Report

Start writing up implementation phase  
Report

Start writing up the experimentation phase  
Report

Start writing up the comparison phase  
Report

+

In progress | 2

Freezing of the cparams and params  
Experiment

Finish problem, goal, scope  
UI

+

Done | 6

Setup Streamlit first page+ tunnelling  
UI

Add videos to the generation screen  
UI

Finish Summarizer Screen  
UI

Create logic for communication between 2 notebooks with atomic writes  
Backend

Explore the possibility of Drawing completion  
Experiment

Create a summary of the report  
Report

+

Appendix A3: The KANBAN board used for the final application and report writing, snapshot image made 16/04/2022.

## 12.2 Appendix B: Risk Assessment

| Problem  | Loss (1-5) | Prob (1-5) | Risk | Plan  |
|--|------------|------------|------|---|
| Colab GPU service platform might become unavailable                              | 5          | 5          | 25   | A different service, PaperSpace, will be utilised instead.  |
| Exams/Christmas period will stop progress for a few weeks                        | 4          | 5          | 20   | This was taken into account when planning. The task that was scheduled during this period was extended.   |
| Illness may slow down the progress   | 4          | 3          | 12   | The time scales for major tasks are overestimated in some parts.  |
| The materials to research might be too difficult                                 | 3          | 4          | 12   | Make notes and attend weekly 1-2-1 meetings with the supervisor.  |
| Minor experience with python might slow down the progress                        | 3          | 3          | 9    | Bringing up any technical issues that I can't solve during supervisor meetings and asking for help allocated a greater amount of time for implementation. |
| Application for Internships/careers related actions might slow down the process. | 3          | 3          | 9    | In October, a major amount of time was spent on sending applications to save time in the later stages of the project.                                     |




---






Appendix B1: The updated risk Assessment table.

---



## 12.3 Appendix C: Image augmentation techniques

| Augmentation technique  | Example image result   | Average CLIP similarity of 4 samples |
|---|--|--------------------------------------|
| RandomPerspective(prob=1)<br>+RandomResizedCrop(prob=1)<br>(used in CLIPDraw) |    | ~0.450                               |
| RandomPerspective(prob=1)   |    | ~0.433                               |
| RandomResizedCrop(prob=1)   |   | ~0.521                               |
| RandomHorizontalFlip(prob=0.5)  |  | ~0.703                               |
| RandomRotation(prob=1)  |  | ~0.426                               |
| RandomAffine(prob=0.8)  |  | ~0.538                               |

|  |  |        |
|--|--|--------|
| GaussianBlur(prob=0.5)   |    | ~0.718 |
| ColorJitter(prob=0.5)  |    | ~0.707 |
| RandomHorizontalFlip(prob=0.5)<br>+RandomAffine(prob=0.4)<br>+RandomResizedCrop(prob=0.75) |   | ~0.491 |
| RandomHorizontalFlip<br>+RandomPerspective   |  | ~0.428 |
| RandomHorizontalFlip<br>+RandomAffine  |  | ~0.418 |

---

Table C1: The effect of the different augmentations on the visual structure of the images.

---

## 12.4 Appendix D: UI Wireframes

A Web Page

https://

Title

Number of lines

Number of iterations

Text prompt if used without chapter

☐ Insert chapter

Submit

This wireframe shows a web page titled "A Web Page" with a browser address bar containing "https://". The main content area includes a "Title" label, two horizontal sliders labeled "Number of lines" and "Number of iterations", a text input field labeled "Text prompt if used without chapter", a checkbox labeled "Insert chapter", and a "Submit" button.

Appendix D1: Wireframe of the home screen when entering text description manually.

A Web Page

https://

Title

Number of lines

Number of iterations

Text area to paste chapter

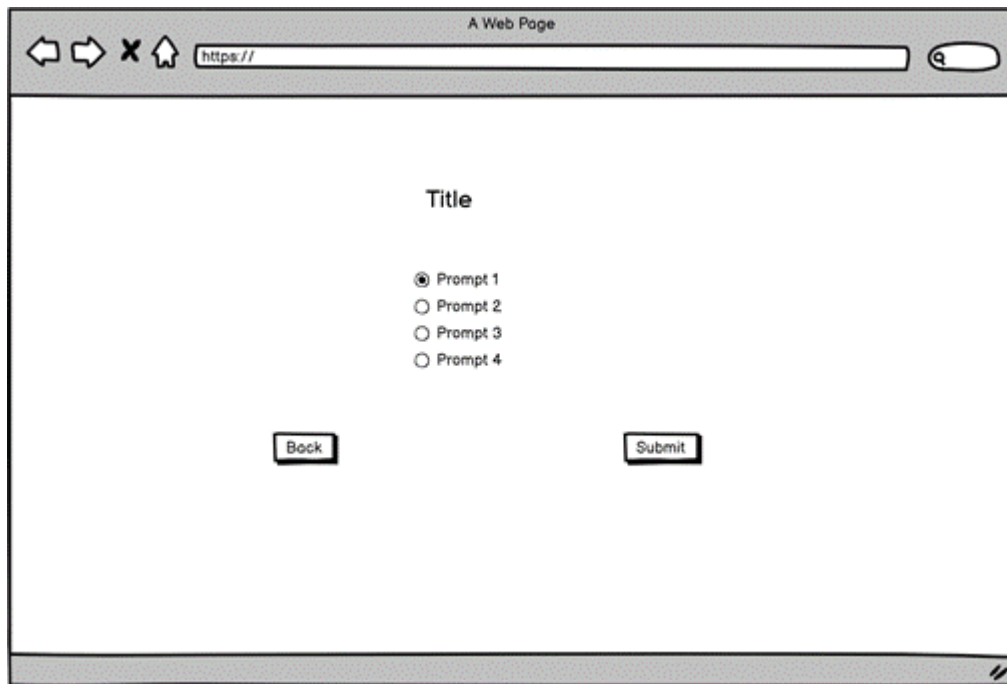
Upload chapter content button

☐ Insert chapter

Submit

This wireframe shows a web page titled "A Web Page" with a browser address bar containing "https://". The main content area includes a "Title" label, two horizontal sliders labeled "Number of lines" and "Number of iterations", a text area labeled "Text area to paste chapter", a button labeled "Upload chapter content button", a checkbox labeled "Insert chapter", and a "Submit" button.

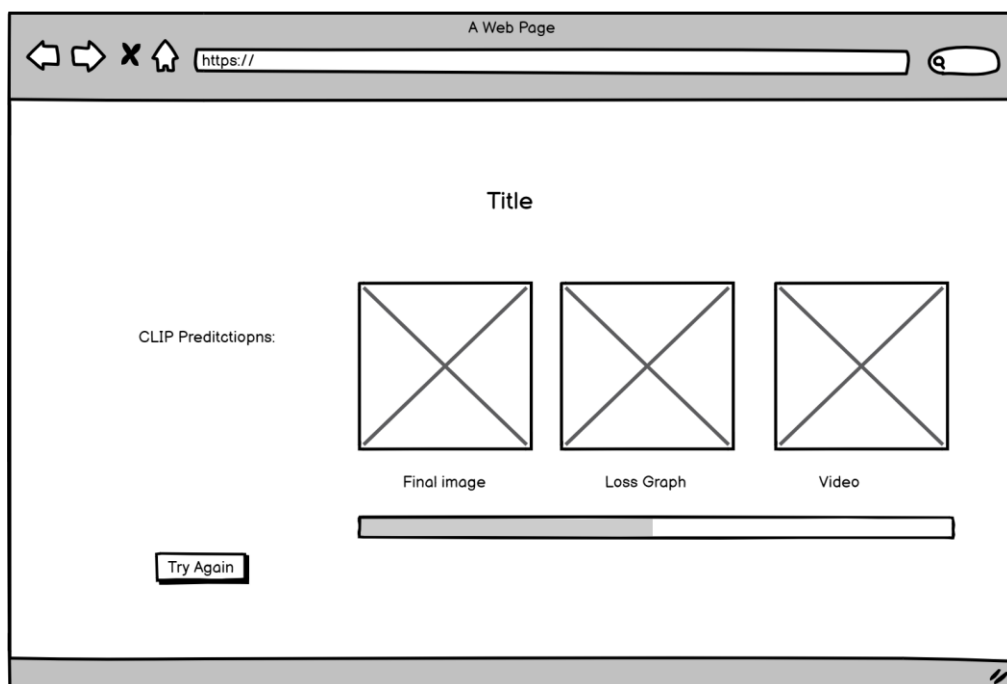
Appendix D2: Wireframe of the home screen when pasting/uploading a chapter text.




---

Appendix D3: The second page that shows the generated headlines if a chapter was entered.

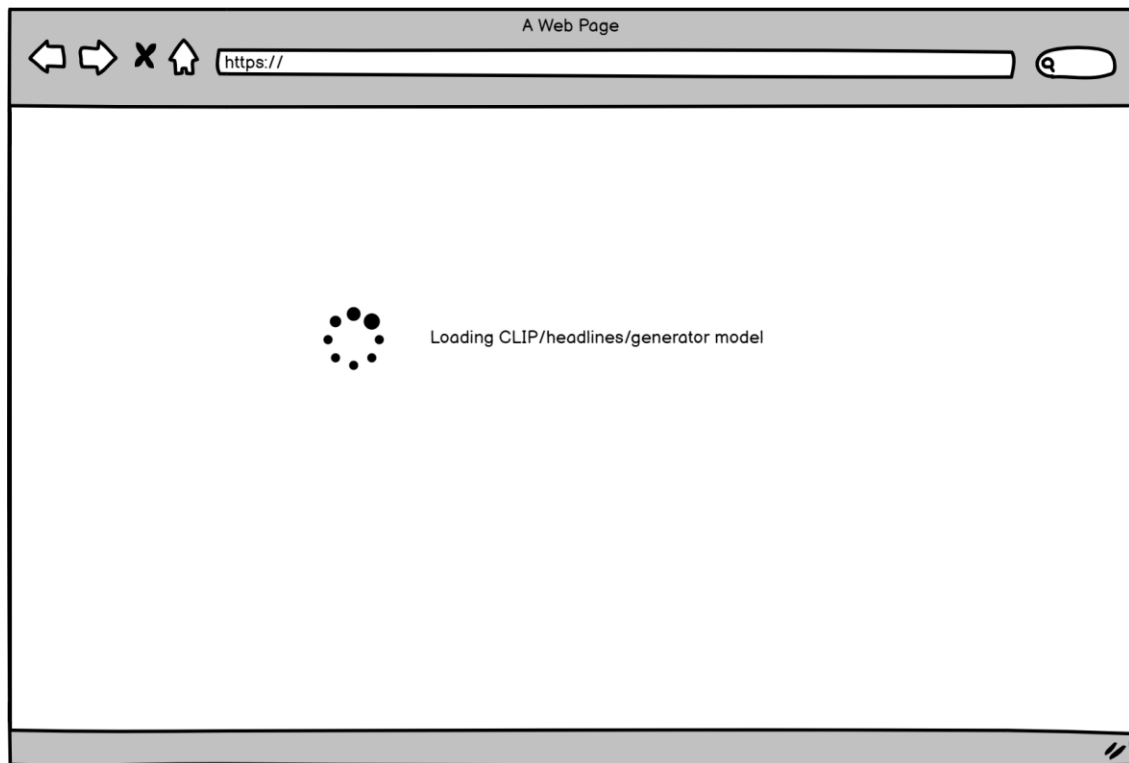
---




---

Appendix D4: The image generation screen.

---



---

Appendix D5: The design of the loading screen used in multiple occasions.

---

## 12.5 Appendix E: Screenshots of the *Streamlit* application

↓

Loading CLIP...

↓

### Text-to-paiting/drawing generation

OpenAI's CLIP + Differential Drawing

Number of lines

100 425 850

Number of iterations

100 400 1000

Insert a text prompt for the drawing here

alice in wonderland 19/100

Submit

☐ Insert book paragraph or chapter

↓

### alice in wonderland

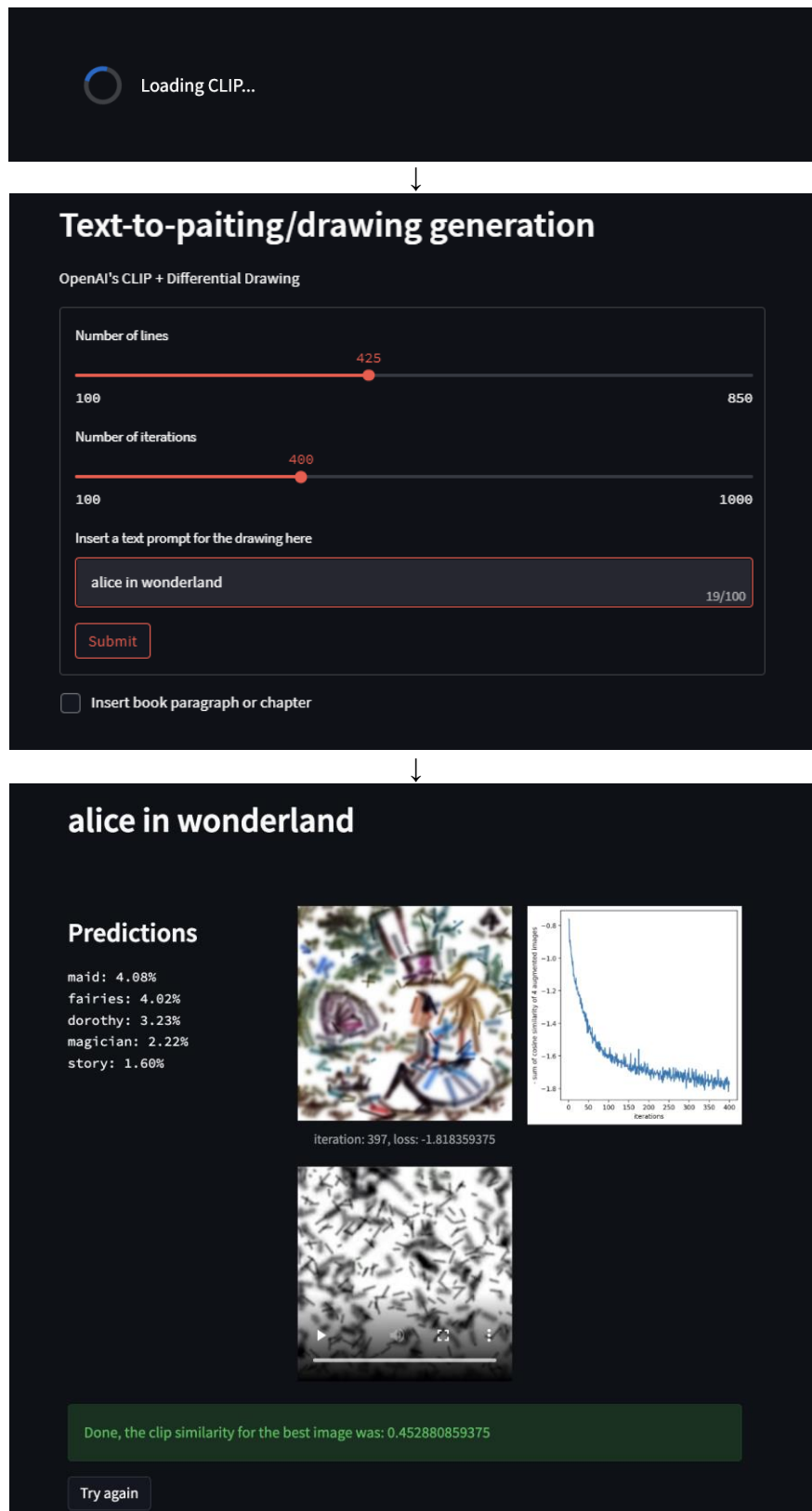
**Predictions**

maid: 4.08%  
fairies: 4.02%  
dorothy: 3.23%  
magician: 2.22%  
story: 1.60%

iteration: 397, loss: -1.818359375

Done, the clip similarity for the best image was: 0.452880859375

Try again



Appendix E1: Steps in the web application to create a drawing from a text description.

## Text-to-paiting/drawing generation

OpenAI's CLIP + Differentiable Drawing and Sketching

Number of lines

100 425 850

Number of iterations

100 400 1000

Paste the chapter contents here...

or upload .txt file

Drag and drop file here  
Limit 200MB per file • TXT

Browse files

Submit

☒ Insert book paragraph or chapter



## LED-booksum + [T5-headline, YAKE, RAKE, KeyBERT]



Generating the prompts from the text...



## LED-booksum + [T5-headline, YAKE, RAKE, KeyBERT]

Choose one of the generated prompts

- ☒ The Boy Who Lives
- ☐ Dudley who is very popular
- ☐ older brother called dudley
- ☐ mr bursley wakes up on monday morning

Submit

Back





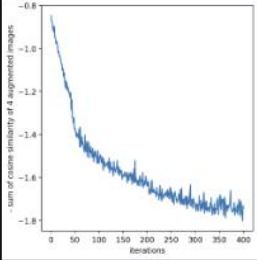
## The Boy Who Lives

### Predictions

boy: 16.65%  
cover: 3.03%  
novel: 2.02%  
son: 1.93%  
show: 1.90%



iteration: 397, loss: -1.80078125



Done, the clip similarity for the best image was: 0.4404296875

Try again

---

Appendix E2: Steps in the web application to create an illustration from a chapter.

---



## 12.6 Appendix F: Examples of the generated images using CLIP+DDS



“a watercolour painting of a cat”  
425 lines



“a cubist painting of a cat”  
850 lines



“a 3D rendering of a fantasy world”  
850 lines



“a drawing of a fantasy world”  
425 lines



“a watercolour painting of a castle”  
850 lines



“a realistic photograph of a castle”  
850 lines



“a realistic photograph of a landscape”  
425 lines



“a 3D rendering of a landscape”  
850 lines



“a watercolour painting of a city”  
850 lines



“a realistic photograph of a city”  
850 lines



“a watercolour painting of a beautiful flower”  
850 lines



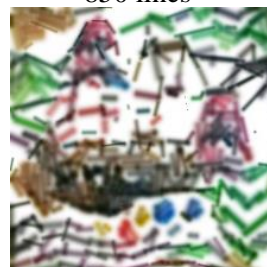
“a beautiful flower”  
850 lines



“a cubist painting of a formula 1 car”  
850 lines



“a watercolour painting of a formula 1 car”  
850 lines



“a watercolour painting of a pirate ship”  
425 lines



“a cubist painting of a pirate ship”  
850 lines



“a medieval empire”  
425 lines



“a medieval empire”  
850 lines



“a realistic  
photograph of a  
modern society”  
850 lines



“a watercolour  
painting of a modern  
society”  
850 lines



“a watercolour  
painting of  
happiness”  
425 lines



“an impressionist  
painting of  
happiness”  
850 lines



“an impressionist  
painting of love”  
850 lines



“a watercolour  
painting of love”  
425 lines

---

Appendix F1: Cherrypicked samples of generated images using the new algorithm for the comparison process.

---





“The prince of Sybaris”



“The Happiest Man in the History of Mankind”



“ms dashwood’s wedding ring”



“The Christmas Vacation in Cleveland, Ohio”



“wealth goes to his first son”



“manor called allham hall”



“hair in each of her fingers”



“Elenor’s Unhappy Marriage”



“finds out she is not in danger”



“Lucy cannot believe that they are goodnatured”



“Wollstonecraft’s Armies”



“Fortify Your Town Again”

---

Appendix F2: Samples of generated images using the NLP pipeline. All images were generated using 850 images.

---

### 12.6.1 Database of the results

The complete database of the images used for comparison and experimentation, together with the results of the NLP pipeline’s runs is available in the archive and at:

<https://docs.google.com/spreadsheets/d/1JChM8Ssfqbv1QeDGvSIDMQXbXcmNupDjySvp1gWxbaM/edit?usp=sharing>

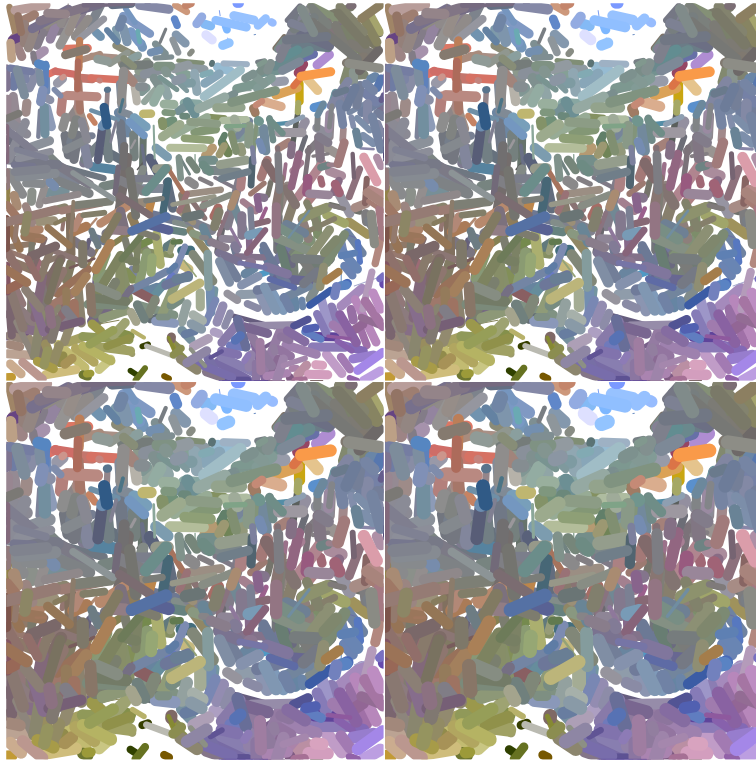
## 12.7 Appendix G: Vector lines widening



---

Appendix G1: The raster image(left) and the vector counterpart before uniformly widening the lines (right).

---

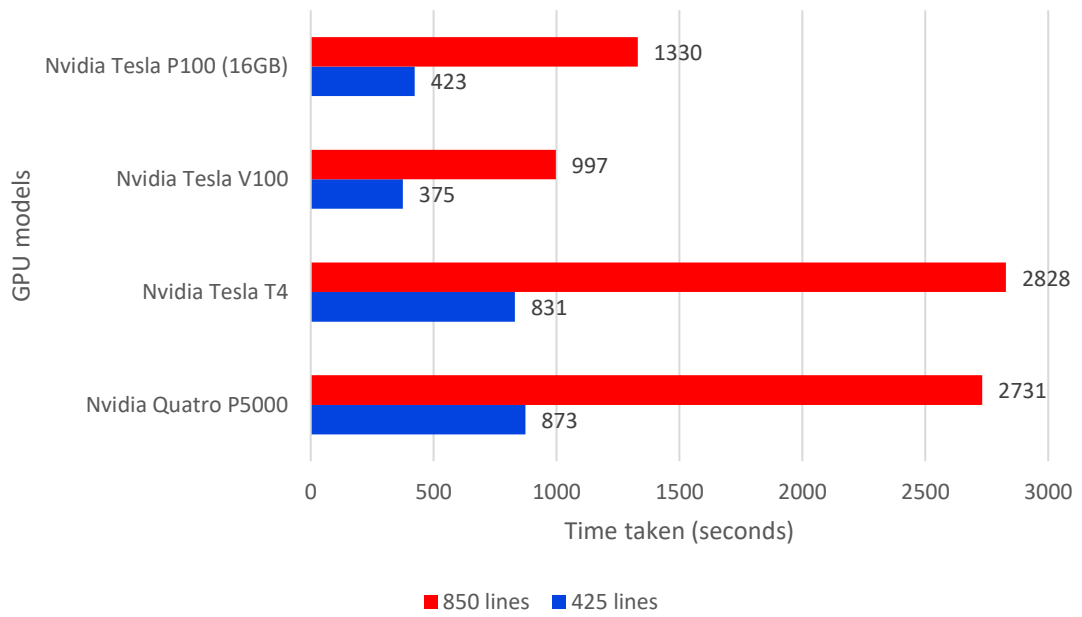


---

Appendix G2: Vector images with line width increased uniformly by 115% (top-left), 135% (top-right), 155% (bottom-left) and 175 (bottom-right).

---

## 12.8 Appendix H: GPU runtimes



---

Appendix H1: Time taken to generate an image with 1000 iterations and 425/850 lines using CLIP+DDS.

---

## 12.9 Appendix I: Original project brief

### AI-assisted generation of art-like images using drawing algorithms and CLIP

**Tomas Mrkva**

**Supervisor: Jonathon Hare**

With the recent advancements in image classification, a neural network from Open AI - CLIP [1] has shown promising results, using its image description capabilities. An interesting application of CLIP, where it was used to help create a text-to-drawing synthesis system (CLIPDraw) [2], has piqued the interest of many with its capabilities. The main idea in this case was not to achieve a real-looking image, but AI-assisted artistic images with the use of strokes in form of curves.

The aim of this project is to apply other drawing algorithms to CLIP, generating sketches/paintings from text and see how they compare to CLIPDraw. Based on the results, there are various applications for this model, such as creating illustrations for a book based on its text. An example of a drawing algorithm that can be used is demonstrated in *Differentiable Drawing and Sketching* [3].

#### The goals of this project:

- Research CLIP and CLIPDraw
- Research the drawing algorithms that could be used
- Implement CLIP together with the drawing algorithms
- Compare the results with CLIPDraw and any other similar algorithms; research whether it can be improved
- Based on the results, research what application would be the best
- Finally, produce a working application of the drawing algorithm working with CLIP (might include further research)

#### The scope of this project:

The scope of this project is not to create a solution to generate drawings/paintings from scratch but rather build on top of the existing research that created CLIP and drawing algorithms to generate drawings/paintings. The complexity of these images will be limited by the available computing resources. The project will mainly focus on relatively simple types of drawings, with more complex ones being introduced depending on the progress. Additionally, it is in the scope of this project to find and implement a suitable application of the images that the model creates.

#### References

- [1] A. Radford et al., "CLIP: Learning Transferable Visual Models From Natural Language Supervision," 2019.
- [2] K. Frans, L. B. Soros, and O. Witkowski, "CLIPDraw: Exploring Text-to-Drawing Synthesis through Language-Image Encoders," CoRR, vol. abs/2106.14843, 2021, [Online]. Available: <https://arxiv.org/abs/2106.14843>
- [3] D. Mihai and J. S. Hare, "Differentiable Drawing and Sketching," CoRR, vol. abs/2103.16194, 2021, [Online]. Available: <https://arxiv.org/abs/2103.16194>