Universiteit Leiden
The Netherlands

Opleiding Informatica

PiCraft

From text to image to Minecraft

Robin Kruijf

Supervisors:
Doughty Hazel & Kaiting Liu

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

15/01/2025

Abstract

PiCraft is an AI tool that generates Minecraft houses. This tool is created to transform Minecraft villages into Minecraft villages based on a text prompt. The tool transforms text into image, image into object and object into Minecraft structure. Using this method the tool transforms each house in the village into a new house based on the user's text description of what type of village it is. An example input is "Bikini Bottom" transforming the houses of the village into the houses of Spongebob, Patrick, and Squidward. The aim of this thesis is to improve the accuracy of text to Minecraft structure generations, by improving the intermediate step of text to image generation. In this thesis three types of datasets with their own fine-tune are compared to find the best performing model for text to image generation based on epochs and prompt. According to the user-analysis performed on 50 participants there is no significant difference between the final structure generated with a text prompt and the final structure generated without a text prompt. However, from the qualitative results it seems that adding a more detailed prompt in the text-to-image stage of the PiCraft tool, results in a better image if the requested data was not originally in the dataset the model was trained on. According to the user-analysis there was no significant difference between the structures generated by different amounts of epochs. By looking at the qualitative results, it seems that the image to 3d stage in the model changes the texture and colors significantly. This results in an incorrect mapping back from color to block. Therefore resulting in roughly the same type of structures. As the tool showed only limited improvements we recommend to explore other methods to generate Minecraft structures.

Contents

1 Introduction

Minecraft is a game where users place cubed blocks to create their own structures. Currently there are a few Minecraft tools that use AI to generate Minecraft structures. None of these AI tools can generate high quality villages based on text prompts. AIbuilds is a tool released January 31 2023, this tool generates Minecraft structures based on the ShapE model[JN23] [Kas]. AIbuilds transforms an object generated by the SHAPE model into a re-sizable structure. The ShapE model is a model created by openAI to transform text into a 3d object. As of late, text based AI models are trained to write code for a Minecraft bot (Elephant.AI) [WS]. This bot places blocks using generated code and therefore is capable of creating complex structures. In summary there are currently two main methods used to create a Minecraft structure. The first method is by using a text to 3d object generation model. The second method is by using a text to code model. Both the AIBuilds tool and the Elephant.AI bot can not generate whole villages with multiple structures from scratch. In this thesis an improved version of a structure generation tool is proposed that can generate villages based on an AI text prompt.

1.1 Thesis Goal

The goal of this thesis is to improve the accuracy of text to Minecraft structure generations, by improving the intermediate step of text to image generation. In this thesis a new structure generation tool is proposed (PiCraft). PiCraft generates villages based on text prompt. PiCraft uses multiple AI models under the hood. In figure 1 you can find the whole layout of PiCraft.

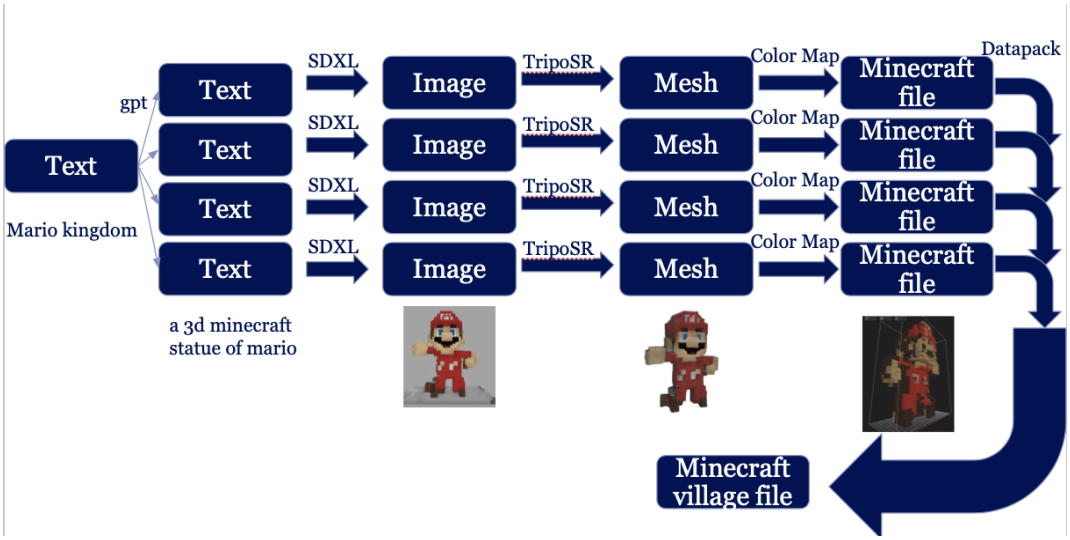

Figure 1: The Pipeline of the PiCraft tool. Arrows indicate the conversion mechanism. Under the 4 blue boxes an example of Mario. Inside the blue boxes the current result.

In this thesis we will look specifically how fine-tuning the text to image part of the tool improves the output of structure generations.
By fine-tuning stable-diffusion XL based on images created by mapping a set of finite colored blocks in Minecraft structures, we hope to: increase accuracy and consistency in block types, identify block-size based on color transition. Therefore, improving the quality of the final structure [DPR23].

The research questions are:
How do Minecraft users rate the buildings as created by PiCraft based on text input?
Does more epochs (updates) of the LoRA fine-tune result in better quality structures?
Does a more detailed text input for the LoRA fine-tune result in better quality structures?

1.2 Thesis Overview

The thesis consists of six sections. The first section describes the related methods of generating Minecraft structures. current used methods for text to 3d models. The second section dives into the inner workings of the pipeline of the PiCraft tool. The third section explains how the PiCraft tool was trained and how the dataset was collected. In the fourth section we describe the experiments and the results of the performed user-analysis. In the Conclusion we answer the research question.

2 PiCraft

As shown in the introduction PiCraft makes use of a few different AI models under the hood to generate datapacks. A datapack is a special type of Minecraft file. A datapack extends Minecraft. The datapack generated by this tool adds Minecraft villages to Minecraft.
In figure 1 you see that the tool first uses ChatGPT to generate four prompts related to the original prompt by using "GPT functions" the four generated prompts each represent a text about a structure that belongs in the village [Ope]. The fine-tuned Stable Diffusion XL model generates a blocked image for each of the text prompts. This image is converted into an object using TripoSR [DTC24]. TripoSR generates an object with vertices and faces from the blocked image. The resulting objects gets voxalised based on the size of the block. Each of the generated structures gets added to a datapack. This datapack automatically spawns in the structures.

2.1 Neural Network

To receive a general understanding of the background of this Thesis we first have to understand what a Neural Network is. Therefore, we first look at a perceptron with only two inputs. In this example we will look at a perceptron that learns the intensity of a pixel in an image.

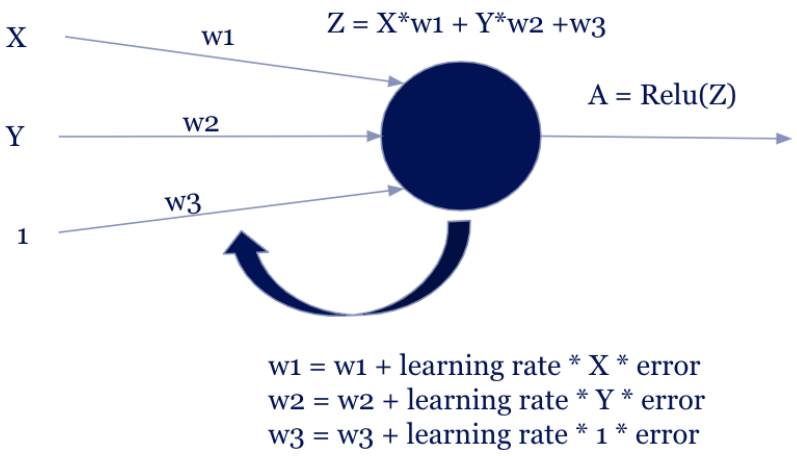

Figure 2: A perceptron with two inputs, a bias node and a single output.

The perceptron in figure 2 has three inputs x, y and a bias node 1. Each of the weights is randomly initialized between 0 and 1. The output of the perceptron is the calculated activation. For calculating the activation we use a single function called Relu.
After calculating how active or bright the pixel is, we can compare our calculated brightness (A) with the brightness from the image at pixel x, y (A0). We can now calculate the error.

error = invRelu(A(0) - A)
learning rate = 0.01

Therefore we can update the weights, training the perceptron on the original image.

The output of the perceptron A can be used as an input for another node in the Neural Network. using this method we can create a multi layered Neural Network.

2.2 Convolutional Neural Network

Using a Perceptron for training on images is inefficient. Instead Stable Diffusion makes use of a Convolutional layers in it's Neural Network [RRO22]. A filter is a matrix of weights. This matrix gets initialised with random values between 0 and 1. By running the network, these weights get updated and become suitable values belonging to a feature. During a convolution a filter is applied in a sliding window fashion over the original image. In stead of learning weights we learn filters, matrices of weights, in a Convolutional Neural Network.

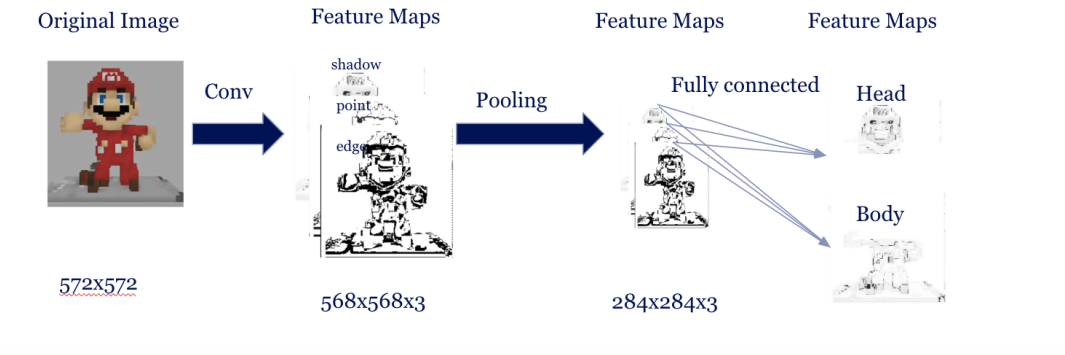

Figure 3: A CNN that takes in an image of Mario and classifies his head and body, using a CNN layer, a pooling layer and a fully connected layer

The Stable Diffusion filter is a 3x3 matrix. Each of these filters learns a different type of feature from the image. In the image there is a filter that learns shadows, one that learns points and one that learns edges from the original image. The 3x3 filter is shifted over all of the pixels in the image. After shifting the filter, over all the pixels, we receive a feature map. The feature map represents features of the original image. The feature map is the filter applied on the image.
To reduce the size of the feature map we convert each 2x2 values in the matrix into a single value. For each of the 4 values we choose the maximum value. This is called max pooling. The x and y of the feature map are now halved. We use a fully connected layer to combine the learned features. After combining the filters, we receive filters that learned the features of head and body of Mario.

2.3 Unet

Stable diffusion makes use of a Unet under the hood. Figure 4 shows an image of the Unet architecture.

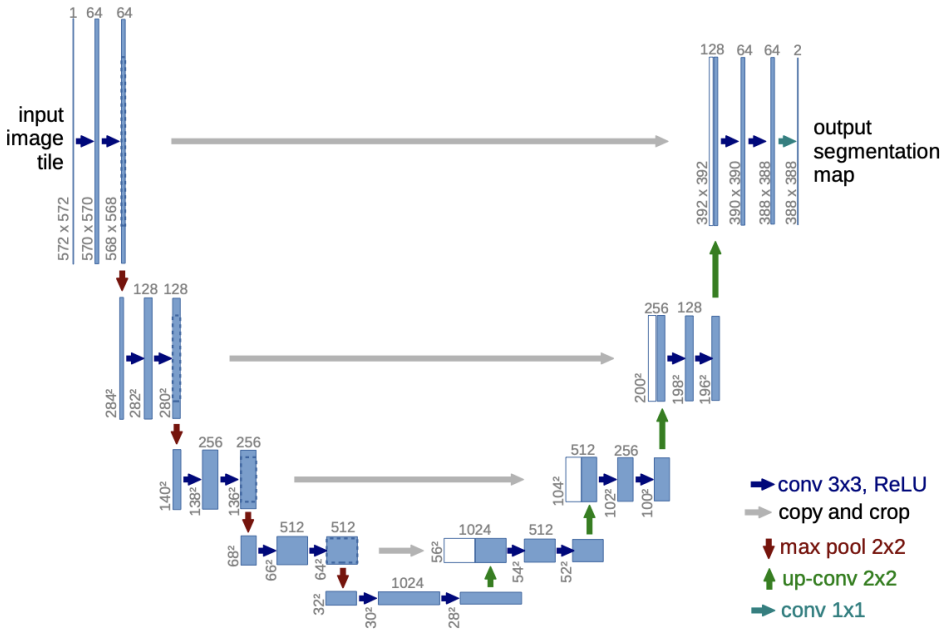

Figure 4: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. [ORB15]

As shown in the image an image is passed into the neural network. The image passed in is converted into a noisy latent representation of the image. As shown in the figure 4, two convolution layers are used. After that max pooling is used to reduce the original image size. This approach is repeated till we have very small images of 32 x 32 pixels. The noise is reduced using max pooling. After receiving the tiny images we use up-sampling the reverse process. We duplicate each pixel to a matrix of 2 by 2. Therefore 28x28 becomes an image of 56x56. After repeating the process of convolutional layers and up-sampling, we are back at the original image size. We now have an image with slightly less noise.

2.4 Diffusion

To train the Unet we start with normal images, we know what the image looks like. We start adding noise to the image. We save each of the images in the noise generation. In figure 5 we add noise to the image. Each of the noisy images is used to train the weights from the Unet.

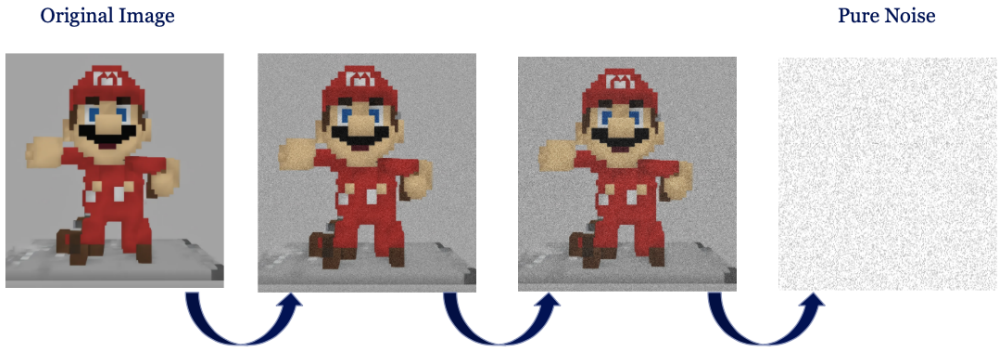

Figure 5: We add noise to the image, each of the noisy images is used to train the weights from the Unet.

To train the model using the image latent, we perform the reverse process. In figure 6 this reverse process is shown.

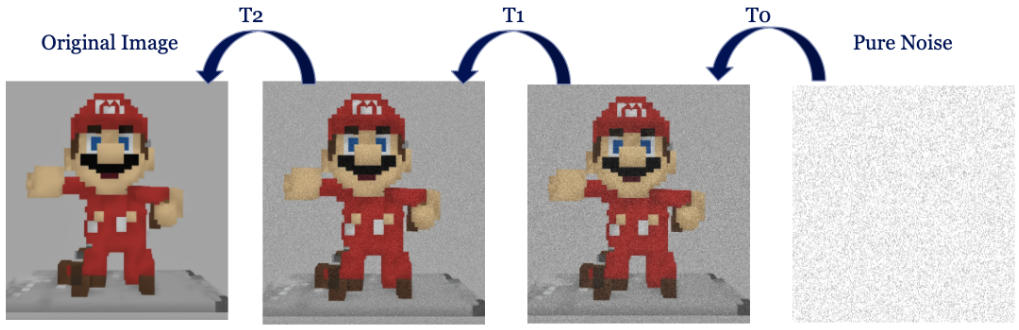

Figure 6: We remove noise from the image, each of the timesteps T represents a full run trough Unet to update the weights.

We remove noise from the image, each of the timesteps T represents a full run trough Unet to update the weights.
This process is repeated till we have an image with no noise anymore.

2.5 Contrastive Language-Image Pre-Training

Now we have a general understanding of the neural network used in Stable Diffusion. Lets dive a bit deeper in how the text is used together with the image. CLIP is a method that generates a vector to represent text and a vector that represents an image. These vectors are referred to as embeddings. In the image below we see how each image is converted into an embedding by the image encoder. Each of the image embeddings is mapped to each of the text embeddings generated by the text encoder. The vectors at the diagonal represent the dot product of the image embedding and the text embedding that belong to each other. The other values represent images and prompts that do not belong to each other. The goal of the model is to have the dot product of the matching prompt and image high and a low dot product for non matching text and images.

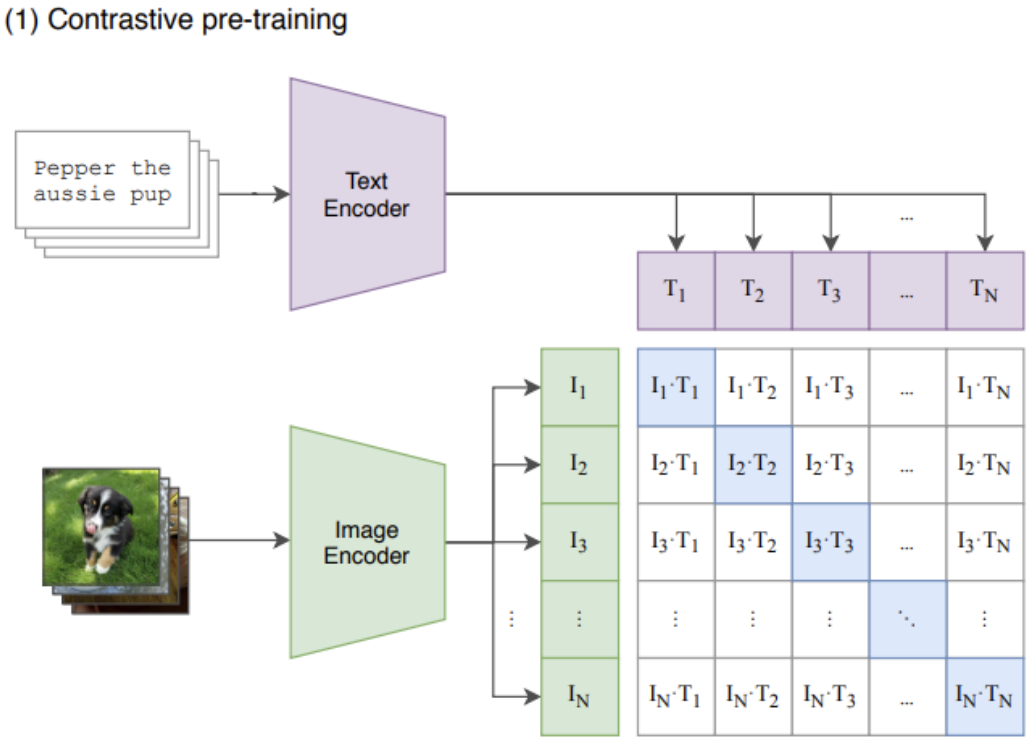

Figure 7: While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes. [ARS21]

2.6 Autoencoder

The image is cropped and only the latent representation of the image (with noise added) is used in the Unet. After removing the noise from the latent image, the latent image is decoded to the original size. The biggest difference between stable diffusion and stable diffusion XL, is how the Auto-encoder works.

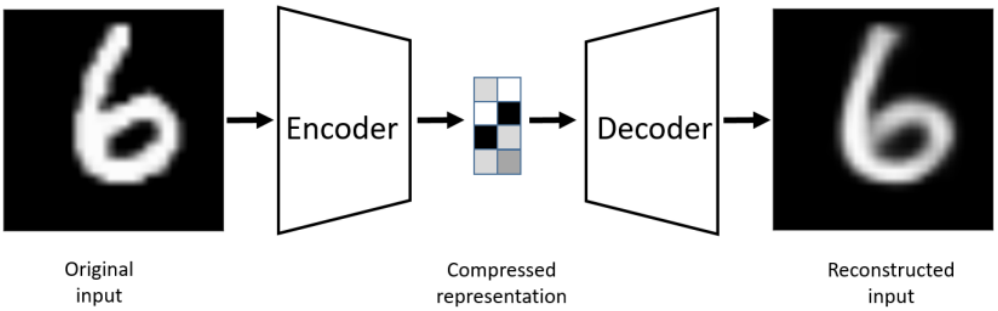

Figure 8: An autoencoder example. The input image is encoded to a compressed representation and then decoded. [DBG21]

2.7 Low Rank Adaptation

Low Rank Addaptation (LoRA) is a method used to fine-tune a model based on reducing large weight matrices by decomposing them into 2 smaller matrices. Therefore, reducing the total amount of weights that have to be trained. Lora is additive and adds a finetuned weight to the original frozen one in the network. The fine tuned weight matrices are small matrices. The dot product of these two smaller matrices is added at each layer of the CNN. During training only the small matrices need to be trained. Therefore the total amount of trainable parameters is far lower compared to the original network. Below an image of LoRA and the small matrices A and B.

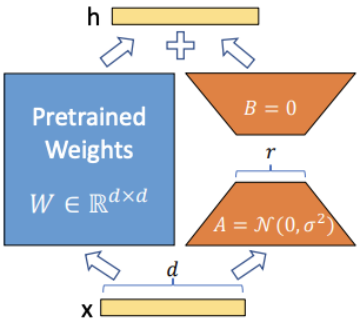

Figure 9: LoRA representation. We only train A and B. [EHC21]

The Lora weights are trained based on epochs. An epoch is a full cycle trough the whole training data set, updating the LoRA weights. By using this approach we reduce the workload to train the dataset.

In this thesis we validate how well LoRA fine-tuning affects the final dataset. Does the fine-tuned version actually improve with more updates?

2.8 Stable Diffusion

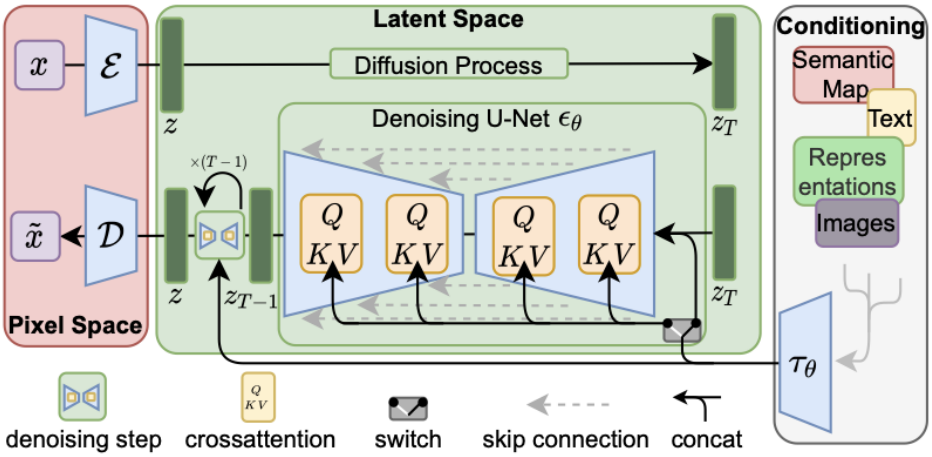

Figure 10: The Stable Diffusion Neural Network [RRO22].

In the pixel space part of figure 10 we see how the auto-encoder transforms an image into latent space x becomes z. During the diffusion process noise gets added to the latent space (zT). In the conditioning part of the image we see how CLIP transforms the embeddings into latent space. This gets added to the noisy image. The noise is removed by repeatedly de-noising the latent space for each time-step T. The auto encoder transforms the latent space back into pixel space, and we receive an output image. [RRO22]

3 Methodology

3.1 Dataset Creation

First, a dataset for SDXL was created. This dataset contained an image of the Minecraft building and a name of the building.

Figure 11: 4 out of 5 images of a rotated windmill in the dataset with its prompt. The image is from dataset 3

The original data comes from a website called minecraft-schematics.com. [lis] This website offers users of Minecraft to upload their Minecraft buildings in the form of schematic files. Users can therefore share their own buildings, they also have to provide a short description of what is in the building. I downloaded 10.000 schematic files with their name.

Each of the downloaded schematic files is converted into a Minecraft building. Each block in the schematic file gets mapped to faces and vertices. Therefore the schematic file is transformed into an object file.

The object is rotated 5 times and an image of the obj is created for each of the rotations as shown in the image above for a windmill.

From the 10.000 schematic files i removed all files that were over 10kb, this reduced the amount of files with large structures. Most of these files had a lot of earth and other noise in them.

3 different sizes of datasets where created.

3.2 Dataset One

3.2.1 Images

The first dataset includes 15.000 images of 3000 buildings. The first 10.000 images are from the outside of the structure, the last 5000 images are from the inside of the structure (all walls removed). The color map of Minecraft block to vertices and faces was perfomed by taking the block id and calculating the average color of that block from the Minecraft game files. Therefore mapping 255 blocks to their according average color.

3.2.2 Prompt

The text prompt used for the first 2000 images started with "a minecraft structure of: " followed by the original description the user gave of the minecraft building. The text prompt used for the 1000 images with only interior started with: "the inside/interior of a minecraft structure of:" followed by the original description of the user.

3.3 Dataset Two

The second dataset includes only the 10.000 images of the outside of the structure from the first dataset using the same prompt.

3.4 Dataset Three

3.4.1 Images

The third dataset includes 8000 images of the outside of the structure.All Minecraft blocks where mapped to 17 colors (not 255).
the colors used are:
1) Red (mc wool): RGB(162, 60, 60)
2) Brown (mc dirt): RGB(124, 90, 50)
3) Brown2 (mc oak planks): RGB(171, 131, 90)
4) Brown3 (mc oak stairs): RGB(139, 99, 92)
5) Orange (mc wool): RGB(225, 136, 60)
6) Green (mc wool): RGB(111, 136, 60)
7) Green2 (mc oak leaves): RGB(66, 136, 60)
8) Yellow (mc wool): RGB(238, 238, 60)
9) Grey (stone): RGB(131, 131, 131)
10) White (glass/white wool): RGB(249, 249, 249)
11) Light Blue (wool): RGB(111, 162, 225)
12) Blue (wool): RGB(60, 85, 187)
13) Black concrete: RGB(0, 0, 0)
14) Magenta (wool): RGB(187, 85, 225)
15) Purple (wool): RGB(136, 72, 187)
16) MC Sand: RGB(255, 223, 127)
17) MC Pink Concrete: RGB(255, 182, 193)

3.4.2 Prompt

The format of the prompt is: <"a Minecraft structure of:"><dataset text><quality type>. An example prompt with:
<dataset text> = Windmill
<quality type> = high quality
Results in the prompt: "a Minecraft structure of: Windmill high quality"

The <dataset text> is the text originally used by the user when they uploaded their building on minecraft-schematics.com.
The <quality type> is the text that I manually added after inspecting the image.
The text prompt for the first 4500 images included either one of the following 4 <quality type>:
- bad low quality
- good quality
- high quality
- a lot of earth

The last 3500 images all included the <quality type>:
- bad low quality

3.5 Color Mapping

For dataset generation: for the first 2 datasets the original colors of Minecraft files where used. This resulted into structures where the color of a torch, which has a lot of empty space resulted in a color close to black. For mapping the blocks to color's the github page from John the Squirrel was used [tS]

For converting the mesh back to a Minecraft file 18 colors where chosen. Instead of the original 255 colors. This was mainly due to incorrect mapping of colors.

For the third dataset I reduced the the amount of colors to 18 where the Euclidean distance between each rgb value had a minimum of 45. Each color also had to represent the original color as good as possible. The colors where generated by chat gpt, I verified that they had an euclidean distance of 45 and where correctly assigned.

In the experiments, the color of a block is determined by the average color of the vertices in the voxel. After the experiments this was changed to the most common vertex color in the voxel.

3.6 Resizing

The size of the mesh is calculated by sorting the vertices on y coords and of looping over the vertices from top to bottom, when there is a change in color it calculates the size of the block. It looks for the mode block sizes and resizes the whole structure based on the mode result.

In later code the resizing was performed by multiplying the original size by 32, instead of relying on calculating the block size. The reason for this was that the map of the block was based on the mode of colors in the block, and not the average anymore. This seemed to be a more reliable method to generate similar sized structures.

3.7 Interior

At each 5 blocks height a floor is added inside the structure. This floor is added only if there is a block somewhere above and somewhere below the current block, but not directly below or above the block. Interior blocks are randomly spawned on the floors. In figure 12 an image of such a floor with interior is shown.

Figure 12: The interior of a Minecraft building

3.8 Replicate

To run the model in the cloud, the model was published on replicate. Now the model can be invoked by API and used by the tool without GPU [Rep].

3.9 Fine-tuning

All 3 of the datasets where used to fine-tune their own model based on Stable Diffusion XL.
Training models is usually a cost expensive task. Fortunately Leiden University offers ALICE [Uni]. ALICE is a cloud server with powerful GPU's. The fine-tuning of the model was be performed on the ALICE platform. Fine-tuning was performed using the created datasets.
For Fine-Tuning SDXL on my datasets an A40 GPU with 160 GB of ram in total and 40 cpu's was used. For fine-tuning SDXL using LoRA I used a learning rate of 1e-04. This learning rate was applied to the text-encoder and the layers in the Unet. The batch size to train the model was set to 1 to reduce the amount of ram usage. This means only a single image was used per iteration. The gradiant-accumulation-step is set to 4. This means the weights are only updated ones every 4 iterations. The effective batch size is gradient-accumulation-step x batch size = 1 x 4 = 4. The code used to fine-tune the model pushes the weights to hugging face ones every 2 epochs. The total amount of epochs for each of the fine-tunes was 3500 epochs. It took the model roughly one week to get to 3500 epochs.

3.10 User Analysis

To answer the research question an user analysis was created. This user analysis consists out of 162 images. The 162 images can be divided into three types of buildings.

1) a statue of Mario
2) Japanese styled houses
3) Sponge Bob his house

Each of the buildings can be divided into: 1000 epochs, 2000 epochs and 3000 epochs
For each text prompt a second text prompt with more detail was created. ChatGPT was asked to generate a more detailed version of the prompt [Ope].

ChatGPT generated the following 3 prompts:
For Mario it generated: a very high quality 3d mesh of a Minecraft: A highly detailed 3D mesh of a Mario statue, styled in the blocky, pixel-art aesthetic of Minecraft. The statue features Mario's iconic red hat with an embroidered "M," a blue overall with golden buttons, a red shirt, white gloves, and brown shoes. The face is meticulously designed, including a pixelated mustache, expressive blocky eyes, and a rounded nose, adhering to Minecraft's voxel-based structure. The proportions remain faithful to Mario's classic design, while maintaining the rigid, cubic geometry characteristic of Minecraft builds. The mesh is optimized for precision with sharp edges, clean voxel alignment, and uniform scaling.

For a Japanese Styled House it generated: a very high quality 3d mesh of a Minecraft: A meticulously crafted 3D mesh of a Japanese-styled house in Minecraft's blocky aesthetic. The design features a multi-tiered pagoda roof with curved, upturned eaves, using dark-colored tiles and wooden beams. The walls are a mix of natural wood and white shoji-like panels, arranged in a symmetrical pattern. The entry includes a detailed sliding door and a stone or wood base foundation. Decorative lanterns, intricate latticework, and wooden railings are incorporated, highlighting traditional Japanese architectural elements. The mesh maintains precise voxel alignment, clean edges, and balanced proportions, optimizing detail while adhering to Minecraft's cubic style.

For Sponge Bob his house it generated: a very high quality 3d mesh of a Minecraft: Create a high-quality 3D mesh of a Minecraft-style house shaped like a giant pineapple. The pineapple has a spiky, blocky top with vibrant green leaves made of individual cubes, and its body is smooth and rounded with a bright orange texture. The house features a large, blocky wooden door at the center and several square windows with a glass-like appearance. The walls of the house should have a blocky, textured surface that simulates the smooth but uneven look of a pineapple's exterior, with a playful, cartoonish design that fits the Minecraft aesthetic.

The users had to rate the structures based on the prompt: "a statue of Mario", "Japanese styled houses", "Sponge Bob his house" and how well the final structures represented the prompts.
For each prompt, dataset and epoch number three structures were created.
The structures where randomly ordered in the questionnaire. Some of the structures did not get in the questionnaire and where therefore left out.

4 Experiments

To analyze the performance of the three different models, a user analysis was performed. A survey was created using google forms. This survey had a total of 140 images. The survey was conducted over 50 participants that play Minecraft. Some of the participants voted a 10 for all of the structures. Therefore the average might be a bit biased upwards.

4.1 Dataset Comparison

In figure 13 the average rating of the three different datasets can be found.
After performing a T-test to compare each 2 datasets with each other, and taking $\alpha = 0.05$. We received the following p values. As null hypothesis we say that each of the datasets is the same.
Dataset 15k compared to dataset 10k resulted in a p value of 0.720. Since $0.05 < 0.720$ we fail to reject the null hypothesis. The difference between the two datasets was not statistically significant.
Dataset 15k compared to dataset 8k resulted in a p value of 0.143. Since $0.05 < 0.143$ we fail to reject the null hypothesis. The difference between the two datasets was not statistically significant.
Dataset 10k compared to dataset 8k resulted in a p value of 0.0476. Since $0.05 > 0.0476$ we reject the null hypothesis. The difference between the two datasets was statistically significant.

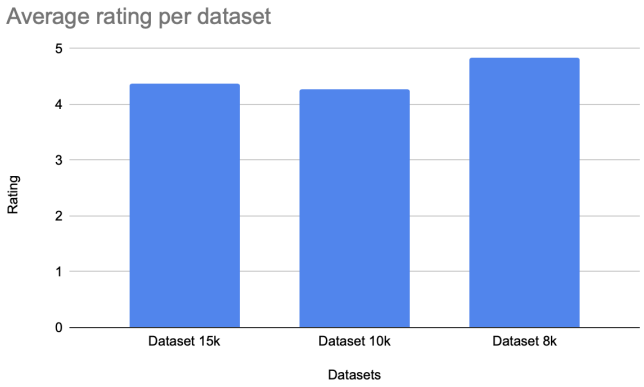

Figure 13: The average rating for each of the different datasets with their results. From left to right: 4.38, 4.27, 4.83

4.2 Qualitative Dataset Comparison

As shown in figure 13, the dataset with 8000 images achieved the best performing results with a slight margin. There are two reasons that can explain this. The reason for this dataset to perform significantly better than the 10k dataset is likely because the text prompt was slightly more detailed. The description of "high quality" houses generally helped with the use of a slightly higher quality. Another reason for this dataset to perform slightly better overall is likely because the dataset was trained using fewer colors. Therefore resulting in output of structures with less colors used. In figure 14 we see three images of Minecraft Japanese styled houses at epoch 3000. Each of the houses has their own according Minecraft structure below it.

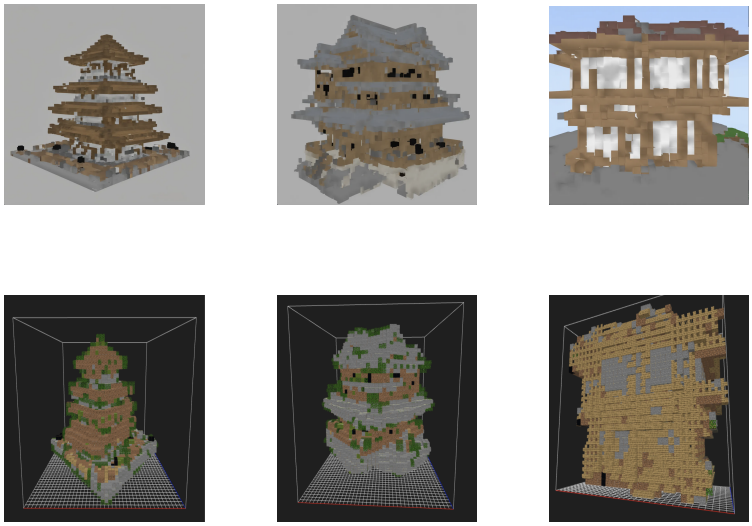

Figure 14: Three Japanese styled houses. The left represents the dataset with 15000 images. The middle images represent the dataset with 10000 images and the right image was generated by the model trained on 8000 images. The bottom 3 images represent how the blocks look after being mapped to Minecraft

As shown, the right most image show most accuracy in using a limited amount of colors for the generation of the building. The resulting accuracy shows higher quality for windows and other simple features. In the left two images black blocks stand for windows, torches and black wool. Comparing the upper three images with the bottom three images, it shows that tripoSR made the original white color look more grey, or darker. This happened to all blocks. The solution to this is explained in the section further improvements.

4.3 Structure Comparison

Below the average rating of the 3 different structures can be found. As shown in figure 15 the Mario statue scored the highest of the 3 structures.

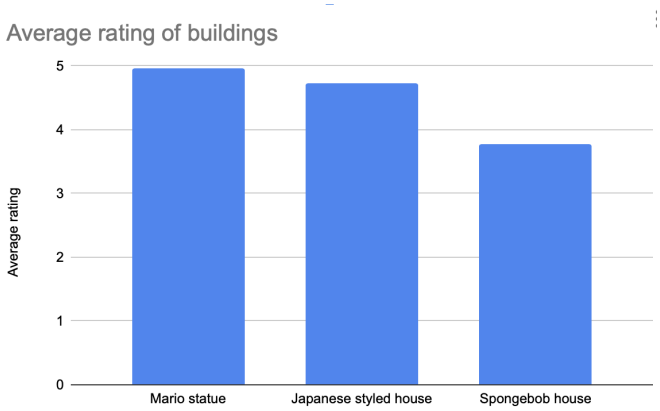

Figure 15: The average rating for each of the different Minecraft structures. From left to right: 4.97, 4.73, 3.77

After performing a T-test to compare each 2 datasets with each other, and taking $\alpha = 0.05$. We received the following p values. As null hypothesis we say that each of the datasets is the same.
For the dataset only consisting out of Mario statues compared to dataset with only Japanese styled houses resulted in a p value of 0.398. Since $0.05 < 0.398$ we fail to reject the null hypothesis. The difference between the two datasets was not statistically significant.
For the dataset only consisting out of Mario statues compared to dataset with only Spongebob his houses resulted in a p value of 0.00012. Since $0.05 > 0.00012$ we reject the null hypothesis. The difference between the two datasets is statistically significant.
For the dataset only consisting out of Japanese styled houses compared to dataset with only Spongebob his houses resulted in a p value of 0.0000039. Since $0.05 > 0.0000039$ we reject the null hypothesis. The difference between the two datasets is statistically significant.

That Spongebob his house performed statisticaly worse than the other generated images is likely due to the base model not having a general understanding of what Spongebob his house should look like. The base model has a general understanding of what the Mario statue and the Japanese styled house should look like. This is likely the reason that the Spongebob house performed a lot worse compared to the other 2.

4.4 Qualitative Structure Comparison

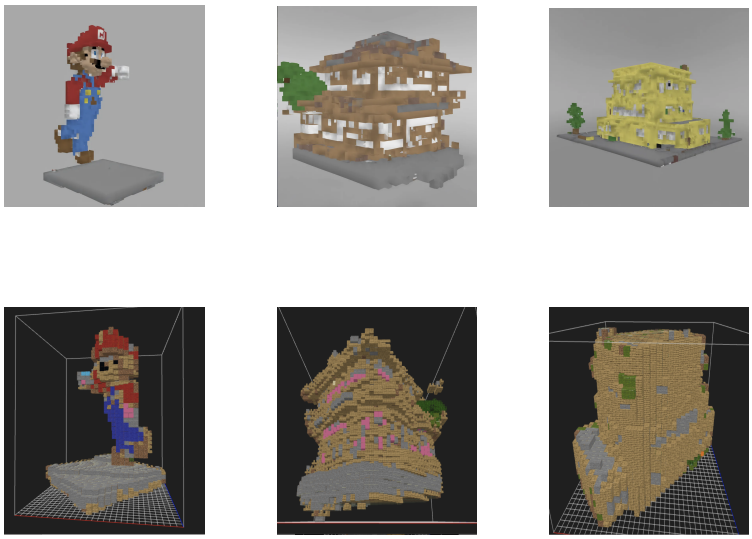

Figure 16: The upper three images where created by the 8000 images model at 3000 epochs. The lower three images where created after Running TripoSR on the images and converting them into MineCraft files.

As shown in the three images above it shows that the model has a decent understanding of how Mario looks. The Japanese styled house looks like it is an actual Japanese styled house. Spongebob his house does not look like a pineapple. Instead it looks like the colors of Spongebob are merged into the house. The ratings was influenced by TripoSR changing the original colors in the image to incorrect colors.

4.5 Prompt Comparison

Figure 17 shows that using an AI generated prompt results in slightly better results.

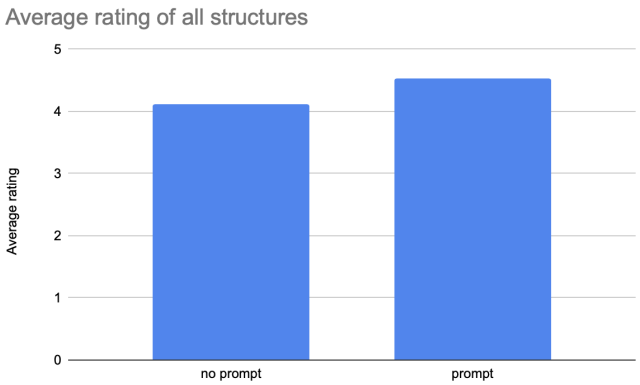

Figure 17: The average rating for using a prompt or using no prompt. From left to right: 4.12, 4.54

After performing a T-test to compare the dataset of no prompt with the dataset with prompt, and taking $\alpha = 0.05$. We received the following p value. As null hypothesis we say that the datasets are the same.
We received a p value of 0.0804. Since $0.05 < 0.0804$ we fail to reject the null hypothesis. The difference between the two datasets was not statistically significant.

4.6 Qualitative Prompt Comparison

Although the difference between the two datasets was not statistically significant. There is very likely an improvement in quality of the images by adding a more detailed prompt. The addition of a more detailed prompt mainly helped with understanding how to add colors. In figure 18 we see how additional information in the prompt effects image generation.

Figure 18: In the left image we see a Mario statue without colors. This is because the image was created after only 1000 epochs. In the right we have the same model with a 1000 epochs, but now a lot of colors have been added. The more detailed description helps the clip embedding to use the correct colors.

4.7 Epoch Comparison

Figure 19 shows that by using more updates the output Minecraft structure received a lower rating.

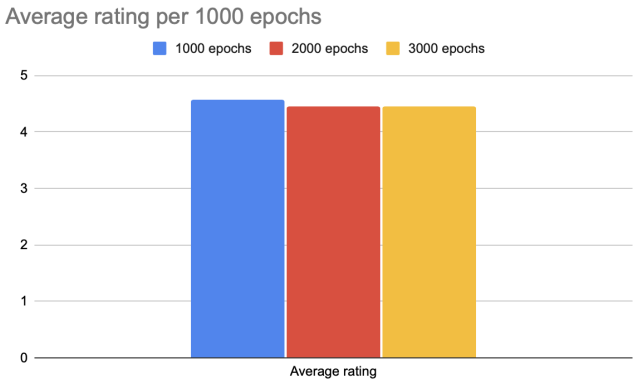

Figure 19: The average rating for the images generated after epochs. From left to right: 4.57, 4.45, 4.45

After performing a T-test to compare each 2 datasets with each other, and taking $\alpha = 0.05$. We received the following p values. As null hypothesis we say that each of the datasets is the same.
For the dataset only consisting out of structures generated by a model at epoch 1000 compared to dataset with structures generated by a model at epoch 2000 resulted in a p value of 0.710. Since $0.05 < 0.710$ we fail to reject the null hypothesis. The difference between the two datasets is not statistically significant.
For the dataset only consisting out of structures generated by a model at epoch 1000 compared to dataset with structures generated by a model at epoch 3000 resulted in a p value of 0.698. Since $0.05 < 0.698$ we fail to reject the null hypothesis. The difference between the two datasets is not statistically significant.
For the dataset only consisting out of structures generated by a model at epoch 2000 compared to dataset with structures generated by a model at epoch 3000 resulted in a p value of 0.983. Since $0.05 < 0.983$ we fail to reject the null hypothesis. The difference between the two datasets is not statistically significant.

4.8 Qualitative Epoch Comparison

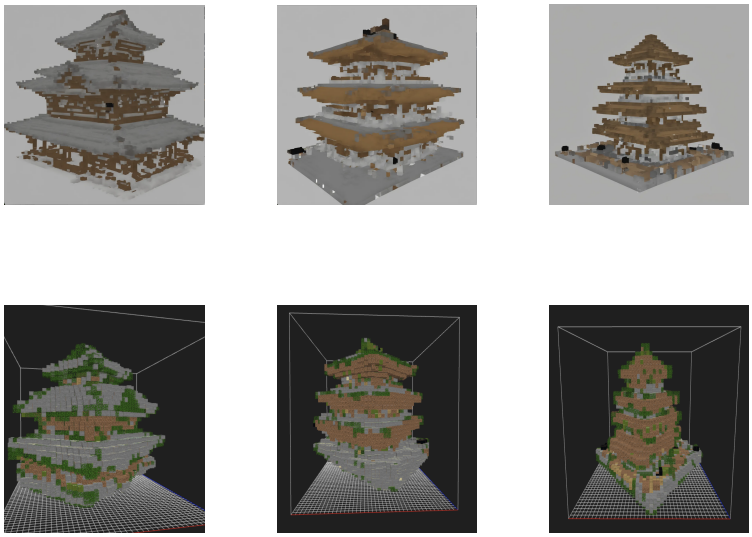

Figure 20: In the left the image of a Japanese styled house from the 15k images dataset generated at epoch 1000. In the middle the image of a Japanese styled house from the 15k images dataset generated at epoch 2000. In the right the image of a Japanese styled house from the 15k images dataset generated at epoch 3000. The bottom three images represent the Minecraft structure after transforming the images from the top.

By looking at the generated images from different epochs in figure 20, we can see that: the basic shape is already available in earlier versions. The fine-tuning mainly adjusts the blocks to be shaped better, over every 1000 epochs. But TripoSR is not capable of adjusting it's generated obj to represent a blocked structure since it was fine-tuned on objects with smooth surface's. Because of this most structures ended up roughly with the same quality as structures with lower amounts of epochs. The mapping of colors by TripoSR is also incorrectly performed. The colors as generated by TripoSR for the object differ from the original image. Therefore block mapping is not as accurate as it should be. The back of the objects generally have a dark shadow over them making the back of structures a lot darker.

4.9 Further Improvements

One of the main issues was TripoSR generating the surface and color of structures incorrectly. Therefore I tried substituting TripoSR with a model from meshy.ai [Mes]. This showed far more accurate results. In figure 21 the comparison of a pagoda object as generated by the TripoSR model from the image. And this object as generated by the model provided by meshy.ai.

Figure 21: In the left the input image. In the middle the generated TripoSR object. In the right the object generated by the Meshy model

As shown in the above three images, TripoSR makes white look a lot darker, more like gray.

4.10 Qualitative Results Training

The model starts of with images from the base model. During fine-tuning the representation of how the model thinks a castle should look like in Minecraft changes over epochs. In the earlier models the shape of blocks was a lot less accurate. An example of this is: how the image of a castle evolved. At the start the minecraft castle has a blue sky and low accuracy in the blocks. After 3200 epochs the castle looks a lot more accurate.

Figure 22: In the left image we see a castle at epoch 0. In the middle we see a castle at epoch 100. On the right we see a castle at epoch 3200.

As shown in the 3 images of figure 22, the castle slowly transforms from an image of a Minecraft castle as created by the base model into a Minecraft castle as fine-tuned by the model. After around a 100 epochs it still uses a blue sky and green grass, as well as that blocks still look inaccurate. After many more epochs the image of the castle transforms into an image without background. The blocks also become a lot more detailed.

5 Conclusion

The aim of this thesis is to improve the accuracy of text to Minecraft structure generations, by improving the intermediate step of text to image generation. In this thesis three types of datasets with their own fine-tune are compared to answer the following three questions:

How do Minecraft users rate the buildings as created by PiCraft based on text input?
It seems that the model trained on the detailed dataset with 8k images performed significantly better than the dataset with 10k images and a lower quality text prompt. A more detailed training prompt helps the model train. The models perform significantly worse on data that it had never seen before such as Spongebob his house.

Does more epochs (updates) of the LoRA fine-tune result in better quality structures?
According to the user-analysis rating, there is also no significant difference in the final structure if more epochs are performed on the fine-tuned version of Stable Diffusion XL. According to the qualitative results it seems that adding more epochs to the image generation process does not improve the overall performance of the PiCraft tool, since TripoSR influences the texture of the generated obj too much.

Does a more detailed text input for the LoRA fine-tune result in better quality structures?
According to the user-analysis rating, there is no significant difference in the final structure if the intermediate image receives a more or less detailed text input. The p value of 0.08 was close to 0.05. So, we might conclude that there is a positive trend towards using a more detailed prompt. This could be verified by performing a user analysis on a larger group than 50 people.

It seems that adding additional information in the prompt helped the image generation process to better understand what type of colors to use in the intermediate image stage. Therefore the final result had a more accurate mapping. By interchanging TripoSR with a model provided by Meshy, buildings are more accurately represented. The final structure has much potential in the field of structure generation in Minecraft. It provides a new way to generate villages. However, as the current tool showed only limited improvements we recommend to explore other methods to generate Minecraft structures.

6 Future

To improve the stable diffusion model in the future, 8k dataset could be made with better text prompts for images. The training dataset still has 17 colors, this could be halved to 9 to provide more accuracy for using correct colors. Instead of using rgb values with a distance of 45 minimum, it could be more useful to train the dataset using the HSV color space. Where the S and V values of HSV are set at 100 percent and there is only change in the H value of the HSV color space. Using this method it might be possible to reduce errors during texturing of the model in the image to object step.

One improvement that could be made on the tool includes the improvement of interior generation. Currently the tool generates a floor at every 5 blocks high in the structure. Instead, the Markov Junior project could be used for the interior design of the Minecraft structures [mxg].

Finally, the speed of generating an nbt file from an object by looping over all the coordinates in the object space and writing the output as text in a file takes a lot of time. This method uses 3 for loops to go trough the whole obj file. It generates roughly 45x45x45 lines of nbt file for a structure of 45x45x45. This should be improved using other methods.

References

[ARS21] Chris Hallacy Aditya Ramesh Gabriel Goh Sandhini Agarwal Girish Sastry Amanda Askell Pamela Mishkin Jack Clark Gretchen Krueger Alec Radford, Jong Wook Kim and Ilya Sutskever. Learning transferable visual models from natural language supervision. *arXiv:2103.00020*, 2021.

[DBG21] Noam Koenigstein Dor Bank and Raja Giryes. Autoencoders. *arXiv:2003.05991*, 2021.

[DPR23] Kyle Lacey Andreas Blattmann Tim Dockhorn Jonas Müller Joe Penna Dustin Podell, Zion English and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv:2307.01952*, 2023.

[DTC24] Zexiang Liu Zixuan Huang Adam Letts Yangguang Li Ding Liang Christian Laforte Varun Jampani Dmitry Tochilkin, David Pankratz and Yan-Pei Cao. Triposr: Fast 3d object reconstruction from a single image. *arXiv:2403.02151*, 2024.

[EHC21] Phillip Wallis Zeyuan Allen-Zhu Yuanzhi Li Shean Wang Lu Wang Edward Hu, Yelong Shen and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv:2106.09685*, 2021.

[JN23] Heewoo Jun and Alex Nichol. Shap·e: Generating conditional 3d implicit functions. *arXiv:2305.02463*, 2023.

[Kas] Kasaisora. Build anything in minecraft using ai. https://m.youtube.com/watch?v=GN4p_oO66ew. Accessed: 25-10-2024.

[lis] listforge.net. minecraft-schematics.com. https://minecraft-schematics.com. Accessed: 25-10-2024.

[Mes] Meshy. Meshy.ai. https://Meshy.ai. Accessed: 15-01-2025.

[mxg] mxgmn. Markovjunior. https://github.com/mxgmn/MarkovJunior. Accessed: 15-01-2025.

[Ope] OpenAI. Chatgpt. https://chatgpt.com. Accessed: 15-01-2025.

[ORB15] Philipp Fischer Olaf Ronneberger and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv:1505.04597*, 2015.

[Rep] Replicate. Replicate. https://Replicate.com. Accessed: 25-10-2024.

[RRO22] Dominik Lorenz Patrick Esser Robin Rombach, Andreas Blattmann and Björn Ommer. High-resolution image synthesis with latent diffusion models. *arXiv:2112.10752*, 2022.

[tS] John the Squirrel. colorstorage. https://github.com/jonthesquirrel/colorstorage/blob/main/README.md. Accessed: 25-10-2024.

[Uni] Leiden University. Alice. https://www.universiteitleiden.nl/en/research/research-facilities/alice-leiden-computer-cluster. Accessed: 25-10-2024.

[WS] jj hunt Wenzhe Shi. Elephantai. https://elefant.gg. Accessed: 14-01-2025.