Synthetic Depth and Semantic Dataset Generation from Minecraft

Evan Goldman
University of California, Riverside
egold018@ucr.edu

Ming Lee
University of California, Riverside
mlee536@ucr.edu

Lauren Gallego Ropero
University of California, Riverside
lgall045@ucr.edu

Abstract

This work investigates the performance of monocular depth estimation (MDE) and segmentation tasks in synthetic Minecraft images, comparing state-of-the-art large-scale models with a custom-trained U-Net[4]. Specifically, we evaluate DepthOfAnything V2[1], a model trained on real-world data for depth estimation, and Segment Anything in Images and Videos (SAM-2), a segmentation model also trained on real-world datasets, against a U-Net trained on over 20,000 Minecraft images. By analyzing performance metrics and visual outputs across synthetic environments, we aim to assess the adaptability of real-data-trained models to synthetic domains and the effectiveness of domain-specific training. The findings contribute to understanding the limitations and potential of large-scale models in synthetic contexts and highlight the impact of domain-focused training on performance.

1 Introduction

Recent advances in self-supervised learning and semantic segmentation models have transformed computer vision. While natural image datasets are abundant, synthetic environments like Minecraft offer a unique opportunity for large-scale labeled data generation. Through injections into the rendering engine, we have the opportunity to effectively generate infinite data.

As humans, we are able to extract relevant information and even understand extremely abstract scenes. Recent foundation models such a SAMv2 and DepthAnythingv2 claim to be able to segment and perform depth estimation on anything on par with a human, so we will verify those claims on our dataset. Our data set is far from realistic, yet still understandable by humans, so it will be a true test of how far these models can go.

The contributions of this work are as follows.

1. Generate a dataset of 20,000 images in Minecraft.
2. Test the performance of SAMv2 on it.
3. Test the performance of DepthAnythingv2 on it.
4. Compare the performance of those models with our own U-Nets trained on the Minecraft dataset to use as a performance benchmark.

2 Minecraft Dataset

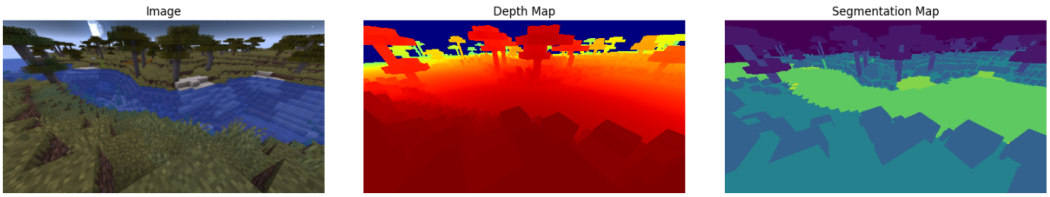

Figure 1: Example of an image in the dataset

To generate this data set, we use the Minecraft Forge modding platform, which uses the Java programming language. Forge gives access to the game files and allows for the addition of extra features, as well as read/write permissions to the host computer. We simply add a class to Minecraft that repeatedly performs the following:

1. Teleports the player's character to a random nearby location in order to give a new scene without the need of world rerendering.
2. Turns the player's camera to a random yaw and a pitch between +45 and -45 degrees from the horizontal plane.
3. Calculates the ray direction in order to perform a raycast at each pixel on the screen. This process is parallelized on the CPU.
4. Save each depth and block to a csv file, along with a screenshot of the scene.

To get the direction of the ray that is needed for a given pixel, we need to cast a 2D screen coordinate to a 3D ray. To do this, we need to use two matrices. First, we need the projection matrix, which is a 4x4 matrix used to convert 3D world coordinates to 2D screen coordinates. Second, we need the view matrix (or the world matrix), which describes how an object is oriented in a 3D environment. We can send the screen coordinates through the inverse of the projection matrix, then the inverse of the view matrix to get our desired ray. Something important to note is that the view matrix is the inverse of the world matrix. Getting the projection matrix was straightforward enough, being accessible from the camera object, but getting the view matrix was very difficult, especially with Forge's API. Forge seems to have inconsistencies with the handedness of their system. It seems some of their calculations use a right-handed coordinate system and some use a left-handed coordinate system, making the view matrix very difficult to construct directly. Luckily, in the calculation of the world matrix, there was no ambiguity in the handedness like cross-products. Inverting the world matrix gives us the view matrix, but since we invert the view matrix later, it is really the world matrix that we want.

We are able to generate about a single image every two seconds with this method. Unfortunately, as this version of Minecraft is written in Java, there is no GPU support for raycasting, which would have sped this process up by many times. The dataset includes both indoor and outdoor scenes, but no underwater scenes. All images are of resolution 2560x1400.

3 Experiments

3.1 DepthAnything v2

DepthAnything V2 is currently one of the best and most efficient MDE models. It is the improved version of DepthAnything[2], an initial effective model that was able to obtain excellent results when analyzing complex scenes but struggles with fine details and transparent objects. These challenges were solved for the second version using a brilliant data generation strategy for combining both purely synthetic images (graphic engines) with unlabeled real images. Why would the use of both types of data provide better results? Here is an analysis of the reasons why using both synthetic and real unlabeled images resulted in a more effective model.

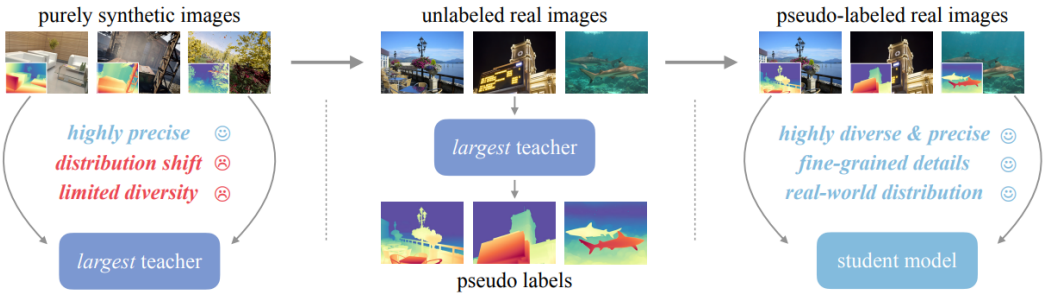

Figure 2: Training architecture of DepthAnything V2.

- **Synthetic data is extremely precise:** Offering a guaranteed lack of noisy depth annotations. This allows the model to get a better understanding of how fine details such as complex curves or holes are illustrated.
- **Cost-effectiveness of synthetic data:** Synthetic data collection is more cost-effective, having a scalable generation that may require additional fieldwork or labor-intensive annotations.

However, synthetic data has a series of intrinsic limitations that can only be overcome using real-world data:

- **Randomness in real data:** Real data has a higher degree of randomness, which is essential for the generalization of the model. Real images have a particular distribution that cannot be perfectly imitated by image generation engines.
- **Scene variety in synthetic images:** Synthetic images have a limited scene variety. Although image generation engines can achieve outstanding precision, they are limited to specific scene domains. The goal is to have a highly versatile model that can perform well under any real image domain.

What model architecture could take advantage of the precise annotations of synthetic images while adapting to a real data distribution that is not limited to specific image domain such as 'landscapes' or 'objects' ?

1. Train a large metric depth estimation (MDE) model using exclusively synthetic data:
   (a) This allows for a high-performance and detail-oriented model that can effectively label new images.
   (b) DepthAnything V2 is trained over 595K synthetic labeled images.
2. Use this model to label unlabeled real images:
   (a) Now, the result is a dataset with not only high precision annotations but that also follows a diverse real-world distribution.
   (b) 62M+ real images were used in the training process of DepthAnything V2.
3. Use this generated hybrid data to train the MDE model.

Currently, three different DepthAnything V2 models have been released (small, base, large). The difference between these models lies in the size of the teacher DINOV2-G[3] model used for the first step of the data generation process. The size of the teacher model has a direct impact on the quality of the real data label generation. In this work, we will look at the performance of all three models in Minecraft images, considering different environments, objects, angles and distances. Additionally, for a better understanding of the model architecture, we will also test the importance of image normalization for these models. This was done by adding some slight modifications to the cloned DepthAnything V2 run.py file, which included suppressing the normalization command and reformatting the image to ensure the output would not suffer any unnecessary alterations.

The aforementioned teacher models are defined by the following two parameters:

- The number of features that describe the embedding of each image patch that is input to the transformer encoders.
- The number of output channels at each of the four training stages, which defines the model's ability to capture complex relationships within the image.

3.1.1 Small

Table 1: Small model output channels and features at each stage

	Output Channels			
Features	**Stage 1**	**Stage 2**	**Stage 3**	**Stage 4**
64	48	96	192	384

The gradual increase in output channels reflects the need for a more balanced model that progressively processes information with increasing complexity. At lower capacities, it's computationally efficient to start with fewer channels and expand them as the model goes deeper, ensuring sufficient capacity at later stages. This model utilizes 24.8M parameters.

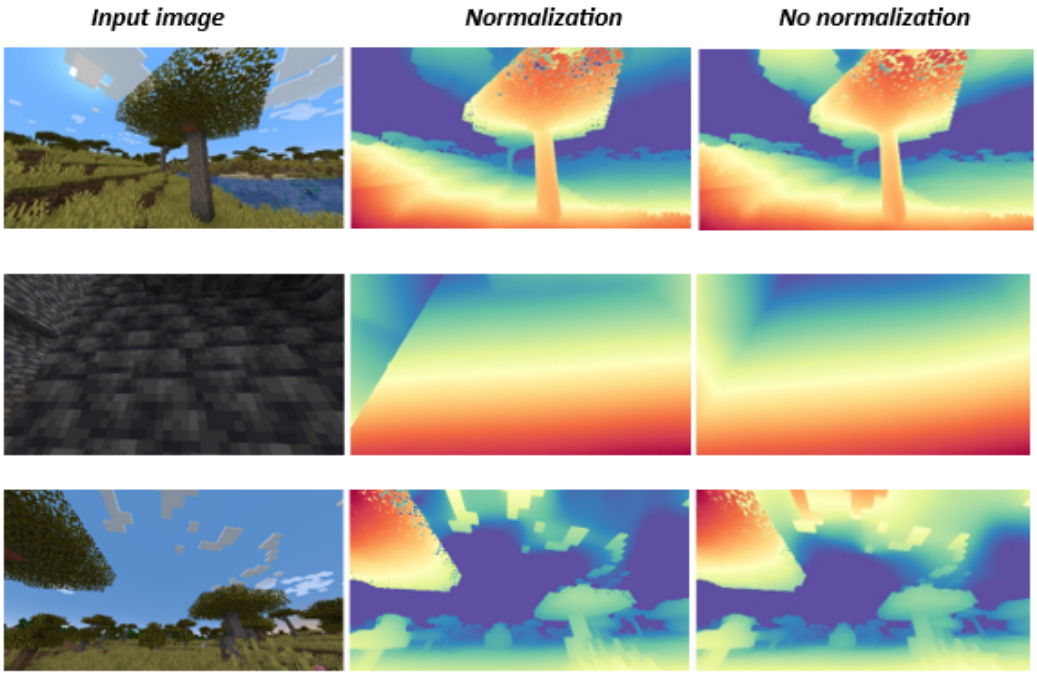

Figure 3: Small Model Experiments

The model can precisely identify the shape of objects like trees and clouds at closer and farther distances. It can effectively distinguish clearly differentiable objects; however, it lacks precision when it comes to identifying details such as grass or edges that are not as noticeable. This phenomenon is reflected in the different height grass blocks. The edges separating different height labels look blurry. By looking at the second image, the effect of not normalized images becomes more obvious. This image tests the model's ability to identify close edges, and it is clear that the normalized image results in a much more precise depth map. Finally, the third image illustrates how normalization also improves model precision when identifying the distance to singular objects around a uniform space such as clouds.

3.1.2 Base

Table 2: Base model output channels and features at each stage

	Output Channels			
Features	**Stage 1**	**Stage 2**	**Stage 3**	**Stage 4**
128	96	192	384	768

The base model uses twice as many embedding features, and slides the stage window to the right, eliminating the first stage of the small model and adding a new and more complex stage. This increases the number of parameters in the model to 97.5 M.

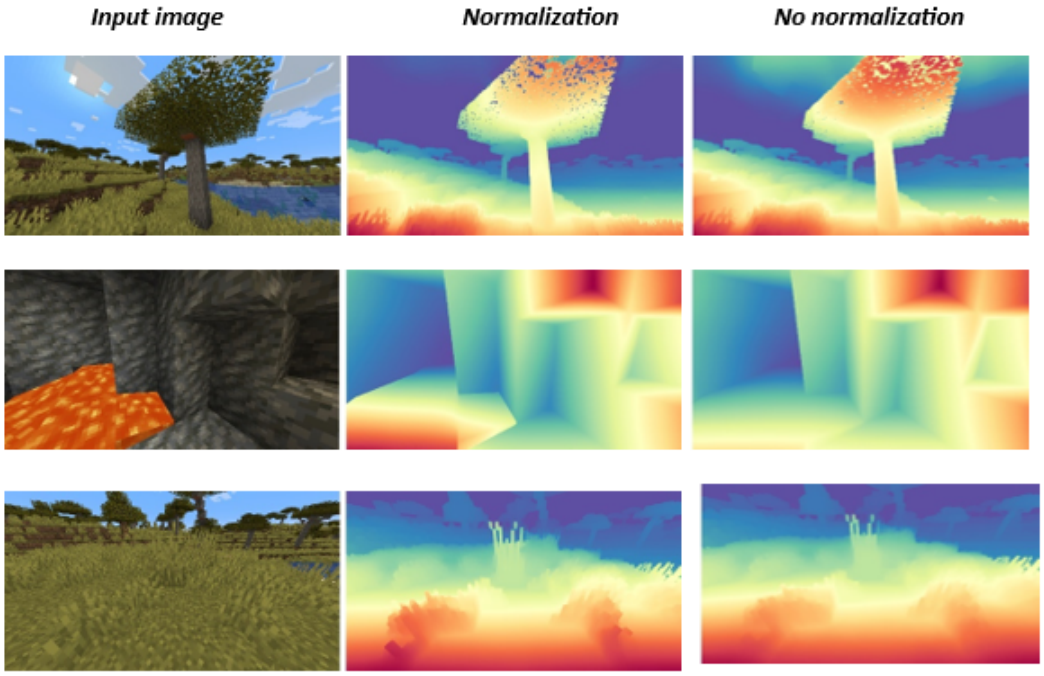

Figure 4: Base Model Experiments

For the base model, we used the same first image to get some insights about if adding twice as many features and output channels would result in more detailed depth maps. Indeed, when comparing the outputs for the small and base models using the first image, not only does the base model capture a larger distance range, but it also draws a more detailed map (see the grass on the bottom left section). However, the normalized base model completely fails to identify the cloud behind the tree in the first image. Eliminating the low-level stage can also have some drawbacks like ignoring farther singular objects. The second image tests the model's performance with two very different textures. In this case, normalization is the key to correctly identifying the 'lava'. The third image is used to emphasize the improvements of the model regarding detail identification, and how the non-normalized image results in a less precise and more blurry depth map.

3.1.3 Large

Table 3: Large model output channels and features at each stage

	Output Channels			
Features	**Stage 1**	**Stage 2**	**Stage 3**	**Stage 4**
384	1536	1536	1536	1536

The large model uses three times as many features as the large model and uses the same number of output channels for all layers. This prioritizes high-capacity processing in all stages.

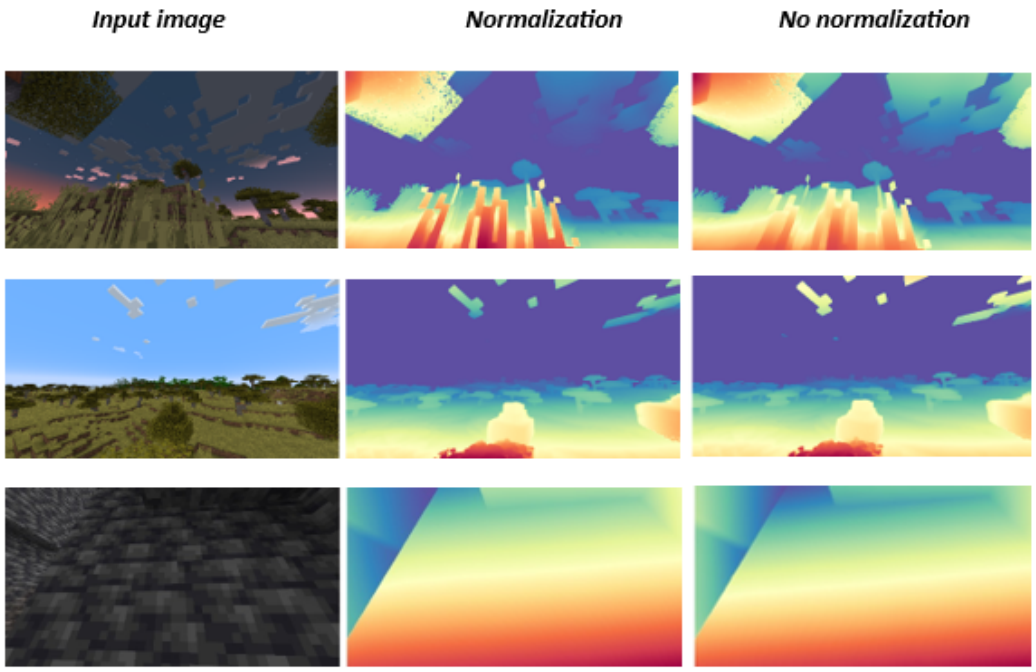

Figure 5: Large Model Experiments

After looking at the first two images, one can already notice the higher precision of the depth maps output by the large encoder. This model can perfectly identify and distinguish textures at closer and farther ranges. The first image tests the model performance with a darker landscape and a very close object. This does not stop the encoder from identifying the rest of the edges in the image, showing that using more features can compensate for the lack of low output channel stages. This though comes with computational cost limitations. The consequences of skipping the image normalization process become less severe in images with the same texture, like the last image, where both outputs are practically the same, and one could even say the rightmost output looks more precise.

3.2 Segment Anything Model v2

Segment Anything in Images and Videos (SAM-v2) builds on the foundation of its predecessor SAM-v1, a model designed for interactive segmentation tasks in images. SAM-v2 introduces capabilities that extend to video, achieving spatiotemporal segmentation. This involves handling individual video frames while tracking object motion and evolving scene dynamics.

SAM-2 resolves the challenges of video segmentation by combining architectural advancements and an innovative data pipeline.

3.2.1 Architectural Highlights

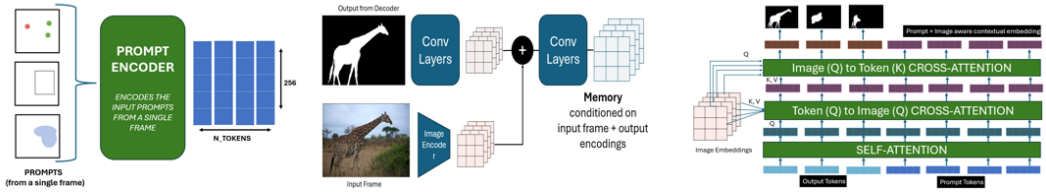

Figure 6: Training architecture of Segment Anything Model v2.

1. **Unified Image and Prompt Encoding:**
   (a) Video frames are encoded independently using a Vision Transformer-based model, generating feature maps that capture spatial properties of the frames.
   (b) Input prompts, such as clicks or bounding boxes, are processed separately by the prompt encoder, transforming them into context-aware representations.
2. **Memory Bank and Memory Attention:**
   (a) SAM v2 incorporates a memory bank that stores embeddings from previous frames, previous input prompts, and object pointers, enabling the model to maintain temporal continuity.
   (b) Memory attention mechanisms integrate past and current frame features to account for dynamic scene changes, object occlusions, and evolving masks.
3. **Efficient Mask Decoding:**
   (a) Similar to SAM v1, SAM v2 uses cross-attention to marry image and prompt embeddings, generating conditional features for segmentation.
   (b) The decoder predicts segmentation masks, Intersection-over-Union (IoU) scores for mask confidence, and occlusion scores to indicate object presence.

3.2.2 Image-Level Segmentation

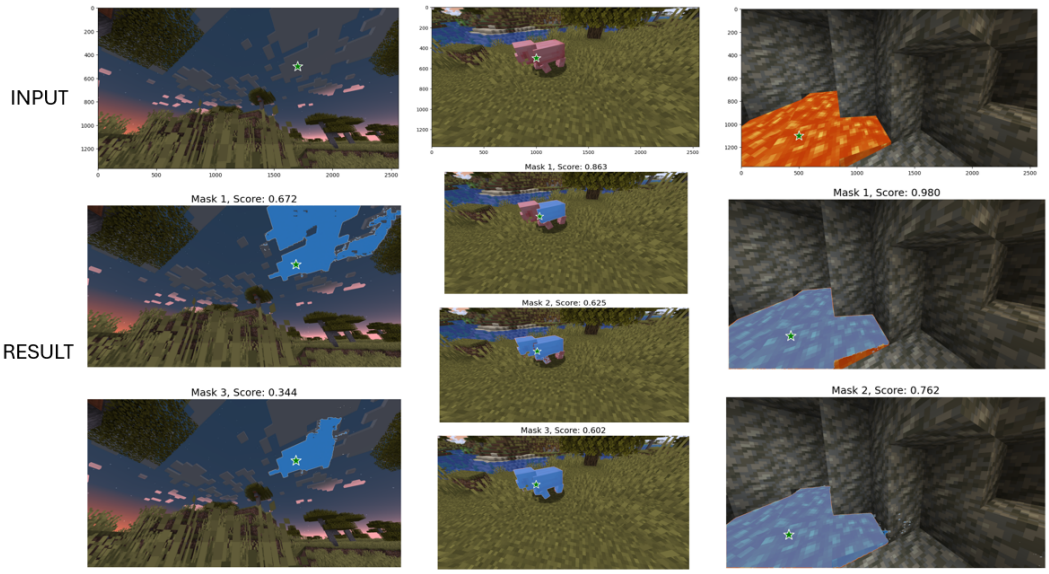

Figure 7: Image result of Segment Anything Model v2.

- Different segmentation masks were generated with varying IoU scores, indicating SAM-V2's confidence in distinguishing objects.
- The segmentation accuracy varied depending on the input prompt type (positive/negative point clicks, bounding boxes, dense masks).
- The positive point marked in green within the image (as shown in the results) served as the reference for evaluating mask prediction.

3.2.3 Video-Level Segmentation

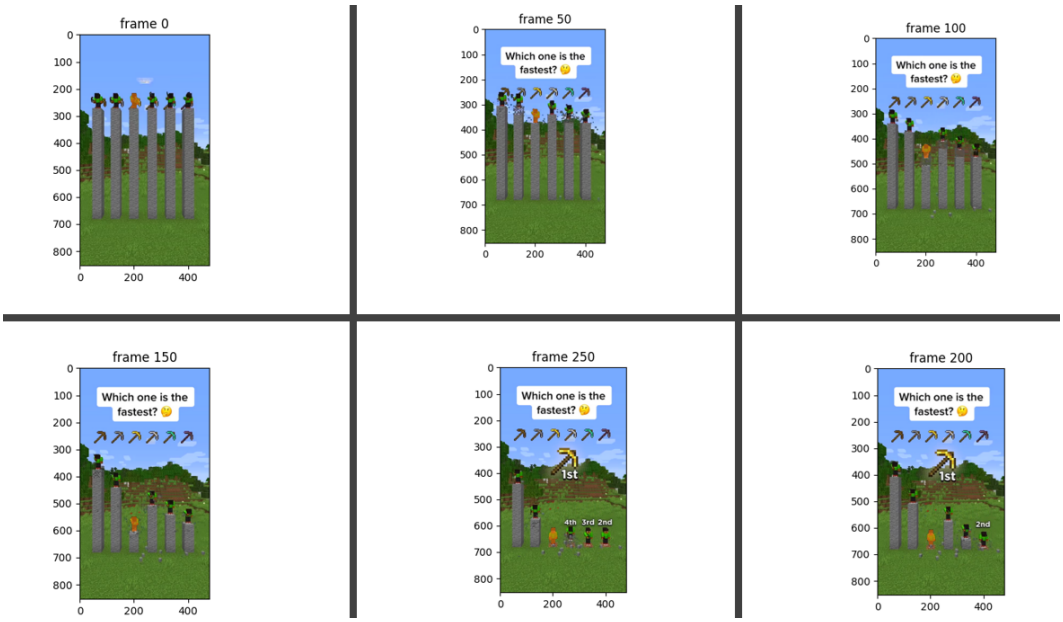

Figure 8: Video result of Segment Anything Model v2.

- SAM-V2 extended its capabilities to handle temporal sequences, tracking objects across frames while maintaining consistency and adapting to occlusions or changes in the scene.
- The memory bank played a critical role in contextualizing frame embeddings with past outputs, ensuring robust segmentation continuity across frames.

3.3 U-Nets

Our U-Net has a depth of 4, with 256 channels in the bottom layer. A depth of 3 caused the model to be very inaccurate, while a depth of 5 resulted in running out of VRAM. We had a tight VRAM constraint even with 24 GB because we were using high resolution images.

We trained the U-Nets like classical Convolutional Neural Networks. For depth estimation, we tried using MSE loss and a weighted MSE loss to prioritize closer blocks. For semantic segmentation, we used Cross Entropy loss. For the optimizer, we used Adam with a learning rate of 0.0001. We train the model for each over 50 epochs on a RTX 4090, resulting in about 50 minutes of training time.

3.3.1 Monocular Depth Estimation

U-Net performs significantly worse than DepthAnythingv2. Although the U-Net attempts to perform metric depth estimation and DepthAnything only performs relative depth estimation, it seems DepthAnything would still outperform U-Nets if they were given the same task. Some things that U-Net struggled with that DepthAnything excelled at are sharp edges and textures. The U-Nets would consistently give the textures at blocks a different depth than the rest of the block. This could be an indicator that an old model such as U-Net is unsuitable for such a task. We are especially

disappointed with the model's performance at estimating depth in nearby objects, and even upon a weighted loss function which prioritizes nearby objects, the model performs just as poorly on close objects, while being even worse on far objects. The model does successfully recognize clouds as very far away, but it creates a gradient on the edges of all areas that are adjacent to the background.

3.3.2 Semantic Segmentation

We are happy with the results of U-Nets on semantic segmentation. They have the general correct idea of what everything is. Something that really impressed us was its ability to relatively accurately identify grass. Grass is very sharp and has lots of holes in its textures, showing the texture of the object behind it. The depth estimation model struggled greatly with grass, but the segmentation model didn't. It does however, also struggle with sharp edges just like the depth estimation model does. We don't expect the model to be able to draw the sharp edges on grass like the ground truth has, but on the other objects, it would have been nice to see some sharp edges.

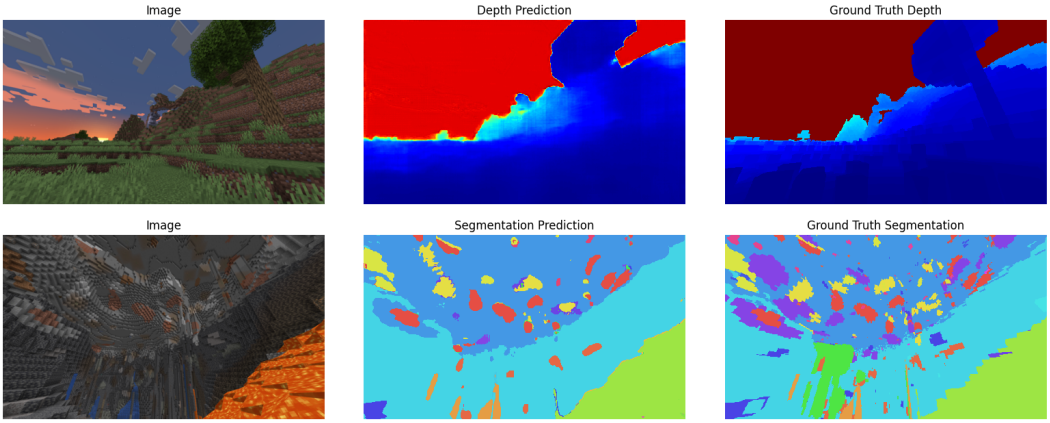

Figure 9: Example of outputs of our trained U-Nets

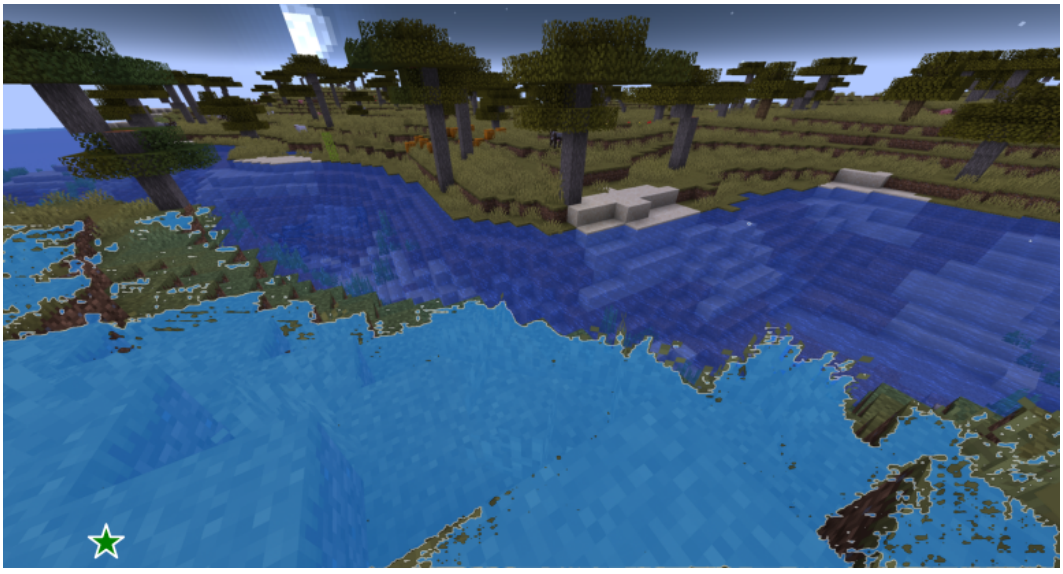

Figure 10: SAM v2 failure on terrain segmentation

4 Conclusion

DepthAnythingv2 was able to achieve impressive performance on our dataset, outperforming our model trained in-domain by far. However, DepthAnythingv2 could not perform metric depth estimation, and we wouldn't expect it to because even humans have a very hard time performing metric depth estimation on Minecraft scenes. Due to the inability of DepthAnythingv2 to perform metric depth estimation, it is unlikely that it could be used for any autonomous agents in Minecraft.

SAMv2 was able to accurately segment objects such as pigs and players pretty well, even in real time while they are moving in a video. They greatly struggled however, with segmenting terrain. Our model was able to segment chunks of terrain far better than SAMv2. Though similar to DepthAnything, the comparison isn't exact since we perform semantic segmentation and SAMv2 performs regular segmentation. It is, however, still able to give us a fair idea about where SAMv2 struggles and could be improved.

Our dataset and the proof-of-concept of generating large scale controllable datasets looks promising in being able to assist new research with creating models. We also hope that through autonomous system development in Minecraft through methods such as reinforcement learning, we could also bridge the gap between computer vision and reinforcement learning, since many autonomous systems need to rely on computer vision, where real time training is not always accessible and simulation environments sometimes take time to set up.

5 Future Work

We notice that the use of vision transformers in reinforcement learning is currently an understudied area. Vision transformers show great promise in being able to capture long range dependencies well, but our current version struggles with sample and computational efficiency, exacerbating the struggles of reinforcement learning. We would like to pretrain a vision transformer backbone on our dataset and then let it loose in a Minecraft reinforcement learning environment to see the performance.

References

[1] Yang, L., Kang, B., Huang, Z., Zhao, Z., Xu, X., Feng, J., and Zhao, H. (2024). Depth Anything V2. *arXiv:2406.09414*, 2024.

[2] Yang, L., Kang, B., Huang, Z., Xu, X., Feng, J., and Zhao, H. (2024). Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data. In *CVPR*, 2024.

[3] Oquab, M., Darcet, T., Moutakanni, T., Vo, H.V., Szafraniec, M., Khalidov, V., et al. (2023). DINOv2: Learning Robust Visual Features without Supervision. *arXiv:2304.07193*, 2023.

[4] Olaf Ronneberger and Philipp Fischer and Thomas Brox. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597*, 2015

[5] Ravi, N., Gabeur, V., Hu, Y. T., Hu, R., Ryali, C., Ma, T., ... & Feichtenhofer, C. (2024). Sam 2: Segment anything in images and videos. arXiv preprint arXiv:2408.00714.