Vision-Based Metric Depth Estimation for Robotic Pruning

Jose Sanchez
Oregon State University
sanchej7@oregonstate.edu

Abstract

Robotic pruning requires accurate 3D understanding of dormant apple tree structure so that a cutting tool can be positioned at the correct cutting location. In this project, we develop approaches for metric depth estimation using synthetic orchard data generated in Blender. A dataset was created consisting of RGB images, dense ground-truth metric depth maps, segmentation masks, and camera poses for dormant Envy apple tree scenes. We first evaluate zero-shot monocular depth models and global linear calibration methods, including a fixed box anchor to provide metric scale in the frame. We also fine-tune foundation monocular depth models such as Depth Anything V2 on the synthetic orchard data. In addition, we investigate stereo RGB-D refinement approaches that use paired camera views and predicted depth maps to refine metric depth estimates. The shared CNN stereo U-Net provided a useful baseline when dealing with noisy and unreliable input depth maps. A frozen DINOv2 model with a depth-side CNN branch gave the best result when input depth came from the fine-tuned DA2 model. Overall, fine-tuning Depth Anything V2 with trunk-masked synthetic supervision provided the strongest single-view result, achieving a validation RMSE of 0.0598 m on trunk depth and 0.0550 m on full-tree depth. The lowest full-tree RMSE came from the DINO RGB + RGB-D refinement model using DA2-fine-tuned depth and 3 stereo pairs as input, achieving 0.0445 m. These results suggest that targeted synthetic fine-tuning is more effective than zero-shot prediction, post-processing calibration, or stereo refinement alone. They also show that pretrained RGB encoders can improve metric depth refinement when the input depth signal is accurate enough, but the added multi-view processing cost must be considered for robotic pruning applications where fast depth estimation is important.

1. Introduction

Pruning apple orchards is essential for crop productivity and tree health. This task requires a specialized skill set, including the ability to identify branches that need pruning with high speed, accuracy, and motor skills to make the cuts at 45 degree angle, which only come with experience [6]. This makes pruning highly labor intensive and difficult to fill due to labor shortages. It also contributes to its high labor cost, making up around 25% of labor costs [3, 6]. In dormant pruning, the removal of selected branches while the tree is not actively growing is consider for robotic pruning. Since the absence of leaves during this state makes the branch structure fully visible, which makes it practical for a computer vision application for a robotic pruner [6].

Prior work has made advancements in using synthetic data from Blender to support real-world orchard tasks such as semantic and instance segmentation [2, 10]. For this capstone, I focused on dormant tree pruning and metric depth estimation. My contribution is a pipeline that is able to generate synthetic camera frames from 100 tree models. The focus on using synthetic data from Blender comes from the lack of real-world dense depth data and the difficulty of retrieving accurate depth from sensors in outdoor orchard settings [13]. This project uses Blender to provide per-pixel ground-truth metric depth maps, segmentation masks, and camera poses [2].

We use the latest monocular depth models as baselines for metric depth estimation. I tested models such as PatchRefineOnce for zero-shot monocular depth, Depth Anything V3 Relative and Metric, and a fine-tuned Depth Anything V2 model [4,5,11]. I also tested tiling and preprocessing approaches to improve the quality of the predicted depth maps before calibration [4]. After that, I used linear calibration to align the predicted depth from these models to the ground-truth depth [2, 4, 5, 11]. Since calibration with trunk pixels depends on ground truth that would not be available in the field, I also tested whether a physical anchor object in the frame could be used as a signal for metric scale. Beyond calibration, I tested whether synthetic fine-tuning could provide a stronger signal. I fine-tuned Depth Anything V2 using masked synthetic supervision and tested different training signals, including trunk masks, box masks, and a box-anchor loss for full tree depth estimation [11]. Finally, I tested stereo and multi-view refinement models to see whether multiple camera views could im-

prove depth predictions. This included a shared CNN stereo U-Net baseline and a DINOv2 l with a trainable depth side branch [7, 8]. These experiments compare whether better metric depth comes from calibration, fine-tuning, or multi-view refinement.

Since my project focuses on metric depth for pruning, the related work falls into three areas: orchard perception, monocular depth estimation, and refinement models. For robotic pruning, the challenge is not just detecting a tree in the image. It requires identifying branches, understanding their three-dimensional positions, and planning precise cuts [6]. Previous research has addressed these challenges through segmentation and skeletonization. For example, Wang et al. introduced a dataset for semantic and instance segmentation in modern fruit orchards using both real and synthetic data [10]. This approach is valuable because orchard scenes often contain thin branches, clutter, and spurs that are difficult to annotate manually. The relevance to my project lies in demonstrating the utility of synthetic orchard data. However, segmentation and skeletonization do not provide metric depth [10, 12]. This matters because the cutting tool must move to a real physical location on the branch, not just a pixel in the image.

Monocular metric depth estimation in agricultural environments represents another relevant area. The OrchardDepth study demonstrates that depth models which perform well on standard outdoor benchmarks may not generalize effectively to orchard settings [13]. This is significant because dormant trees present unique challenges: branches are thin, there are numerous gaps between structures, and a substantial portion of the image may consist of empty background. In my work, I employed recent monocular depth models such as Depth Anything V2, Depth Anything V3, and PatchRefineOnce as initial baselines [4, 5, 11]. However, I did not assume that their outputs would be sufficiently accurate for pruning applications. Consequently, I evaluated calibration, preprocessing, and fine-tuning strategies.

ZoeDepth is also related due to its focus on metric depth prediction. This model employs adaptive depth bins, allowing it to predict and iteratively refine depth estimates before generating the final output [1]. Although I did not select ZoeDepth as one of my final models, it provides valuable context by illustrating how monocular depth models attempt to transition from relative to metric depth estimation. In my project, a similar challenge arises when comparing zero-shot predictions, calibrated outputs, and fine-tuned results.

The final area of related work concerns model architecture. U-Net style models are advantageous for dense prediction tasks because they encode image features and decode them to produce full-resolution outputs [8]. This is relevant to my use of a shared CNN stereo U-Net as a refinement baseline. Additionally, I evaluated DINOv2, which offers pretrained RGB features, to investigate whether combining a frozen RGB encoder with a trainable depth branch could improve metric depth refinement [7]. These two approaches make up my final evaluations in my project: determining whether metric depth estimation for robotic pruning benefits more from calibration, synthetic fine-tuning, or stereo and multi-view refinement.

1.1. Synthetic Dataset and Problem Formulation

Reliable ground-truth metric depth maps are challenging to obtain in the real world. Sensor data can be noisy in outdoor orchard settings due to lighting, and depth sensors can produce sparse point clouds. Since this project requires dense per-pixel depth output, Blender is used because it gives access to ground-truth depth for every pixel, along with camera pose and segmentation information.

We were provided a dataset of 100 different apple trees, including Envy and UFO tree structures. The tree files consisted of JSON files that contained cylinder information. Each cylinder spans about $\sim$10 cm in length and stores information such as radius, orientation, and centroid coordinates. Each cylinder also has a tag for whether it belongs to the trunk, branch, or spur. This allowed me to control which part of the tree gets rendered and to generate masks for different tree structures. I was also able to apply four bark textures: $bark\_brown$, $bark\_brown\_02$, $willow\_bark$, and $willow\_bark\_02$. From this setup, I could manipulate the camera coordinates and intrinsics, such as focal length and principal point, and record this information for every camera view.

The problem in this project is to estimate a metric depth map from RGB images or multiple nearby RGB views. For each image, the model predicts a dense depth map: $I_{\text{RGB}} \rightarrow D_{\text{pred}}$ where D_{pred} should match the Blender ground-truth depth D_{gt} in meters. For the stereo and multi-view experiments, the model receives multiple nearby RGB views and their corresponding predicted depth maps. The goal is to use the extra views as additional context to refine the depth estimate for the target view. Depending on the experiment, the metric depth map came from either PRO, calibrated PRO, or fine-tuned Depth Anything V2. The main evaluation metric is RMSE in meters over the target mask, such as the trunk mask or full-tree mask.

Much work went into getting the data pipeline right. At first, running scripts inside Blender was too slow. Running them through VS Code helped but was still not fast enough for generating larger datasets and running gated depth models like Depth Anything V3 Metric, Depth Anything V3 Relative, and PatchRefineOnce. To generate larger datasets, I moved the pipeline to Blender 4.2.13 to run on the HPC, especially when multiple views per tree were needed.

My data hierarchy remained consistent across experiments. For each rendered frame, I stored the RGB im-

age, ground-truth depth, camera pose, camera intrinsics, segmentation mask, and predicted depth maps from the single-shot depth models I tested. For trunk masks, I used Blender's object index/render pass information to isolate trunk pixels by setting them to 1 and non-trunk pixels to 0. This allowed training and evaluation only on pruning-relevant pixels instead of letting the background dominate the loss or RMSE.

I generated different types of camera frames depending on the experiment. The trunk-only shots were used for monocular calibration and fine-tuning. These included multiple images along the trunk with controlled camera movement. The shots had varied poses with random offsets from the first cylinder of the tree. For the box experiments, the setup was similar, but I added a box object at a fixed location from the camera. The box was about 30 cm in front of the camera and approximately 15 cm by 7 cm. It was used to test whether a known object in the frame could act as a metric scale anchor.

For stereo and multi-view experiments, I used paired or grouped camera views from nearby positions. This tested whether multiple views could provide more geometric context than a single image. The stereo models used RGB views and predicted depth maps as input, then output refined metric depth maps.

The train and validation split was done by tree rather than by individual image. This is important because images of the same tree are visually similar, especially when captured by nearby cameras. If the same tree appeared in both the train and validation sets, the validation result could appear better than it actually is. Splitting by tree ensures the validation set tests whether the model can generalize to unseen tree structures.

2. Methodology

2.1. Tiling for Higher-Resolution Depth Maps

Depth Anything performs inference at a configurable internal processing resolution rather than at the original frame resolution. By default, the model uses **process_res = 504** and **process_res_method = "upper_bound_resize"**, which resizes the longer frame dimension to 504 while maintaining the aspect ratio. For input frames of 1920 $\times$ 1080, this reduces the image to approximately 504 $\times$ 280 prior to depth prediction. Higher resolutions, such as 756 and 1024, are feasible but require increased memory and runtime. Initial testing was conducted on an Ubuntu laptop equipped with an Intel Core i9-14900HX, an NVIDIA RTX 4070 Laptop GPU with 8 GB VRAM, and 32 GB RAM. Due to these hardware constraints, the largest model was used with the default output resolution, resulting in depth outputs generated in approximately 9 to 11 seconds per image.

This resolution limitation motivated the tiling approach. The initial workflow was adapted from the open-source *TilingZoeDepth* implementation [9], which used the ZoeDepth monocular depth model by Bhat et al. [1]. The process starts by producing a base depth map at the model resolution, about 504 $\times$ 280, to capture the global scene structure. Then the original RGB image is split into overlapping tiles. Each tile is processed independently by the monocular depth model. The tile depth maps are stitched back together using a gradient mask and weighted averaging to produce a higher-resolution depth map, as shown in Fig. 2.

This process is repeated with smaller tiles, enabling local regions to utilize more of the model's effective resolution. This approach is beneficial because thin branches may be lost when the entire image is resized to a significantly lower resolution. In the final stage of the algorithm, the base, medium-resolution, and high-resolution depth maps are combined using edge masks derived from the original RGB image. A blurred edge filter is applied to the high-resolution depth map, and the masked high-resolution result is blended with the average of the low- and medium-resolution depth maps. The final output of this tiling algorithm is presented in Fig. 3, while model outputs without this preprocessing step are shown in Fig. 1.

Figure 1 presents a comparison of depth map outputs from Depth Anything V3 Metric (DA3 Metric), Depth Anything V3 Relative (DA3 Relative), and PatchRefineOnce (PRO). This comparison was used in the initial evaluation of each model's ability to preserve tree structure and fine-branch detail. The primary drawback of the tiling approach was increased runtime. With 50% tile overlap, the method required approximately 300 inference calls per frame, resulting in about 2 minutes of processing per frame on the HPC, excluding model loading. To reduce computational cost, lower overlap values, such as 10%, were also tested. This consideration motivated the adoption of PRO, which can produce high-fidelity depth maps using Depth Anything V2 as a backbone in approximately 11 seconds per frame.

2.2. Linear Calibration Baseline

Monocular depth models such as DA3 Metric, DA3 Relative, and PRO output depth maps at different numeric scales. Since the metric ground truth is available from Blender, the key idea was to test whether an optimal scale factor α and offset β could align the prediction with the ground truth without retraining the models. For each pixel, this is written as:

$$D_{\text{gt}} \approx \alpha D_{\text{pred}} + \beta. \tag{1}$$

Calibration was performed only on tree pixels because the trunk accounted for less than 1% of the frame, and I did not want background artifacts to influence the calibration.

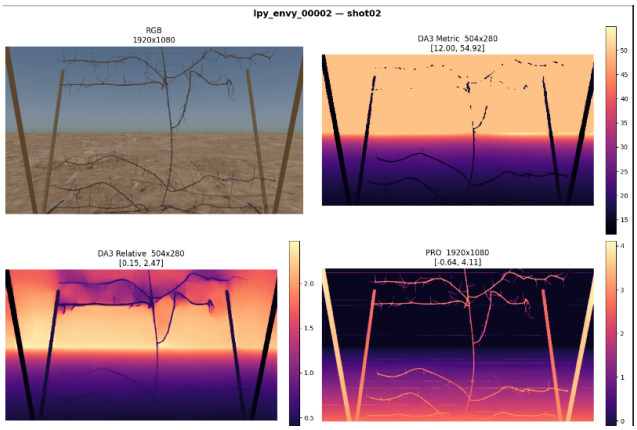

Figure 1. Monocular depth model outputs with their predicted depth ranges.

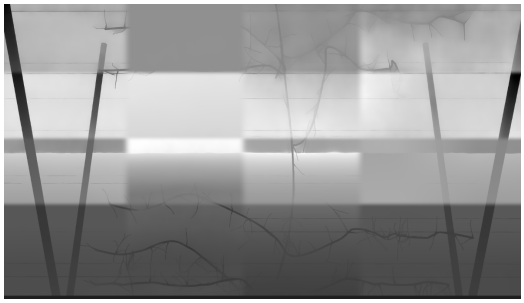

Figure 2. First pass result at medium resolution.

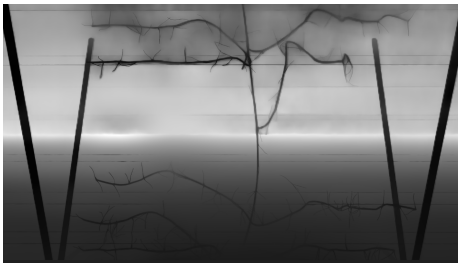

Figure 3. Final map combining base, medium-resolution, and high-resolution tiling depth passes.

In the first version of the dataset, calibration was done on fixed camera poses across different trees. About 80% of the data was used to fit α and β, and the remaining 20% was used for validation.

When scaling to a larger dataset with over 24,000 frames, the use of **np.linalg.lstsq** resulted in memory issues due to the need to store every valid pixel. To address this, a streaming least-squares approach was implemented. Rather than storing all pixels, the method accumulated the number of pixels, the sum of predicted depth values x_i, the sum of squared predictions, the sum of $x_i y_i$, and the sum of ground-truth depth values y_i. Here, x_i denotes the model prediction and y_i represents the masked ground-truth depth. The depth maps were flattened to allow incremental processing of valid pixels.

$$\begin{bmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & n \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \sum x_i y_i \\ \sum y_i \end{bmatrix}$$

This approach enabled calibration without the need to store all pixels in memory. One drawback is numerical instability, as the sums are computed over millions of pixels. This issue was mitigated by employing double precision and mean-centering, which reduces rounding error by working with each pixel's deviation from the mean rather than summing large prediction values.Per-model calibration and best-preprocessing results for both datasets are reported in Tabs. 2 and 9.

2.3. Preprocessing Sweep

Simple preprocessing techniques were evaluated to determine whether they improved calibration results. The motivation was that some monocular depth outputs exhibited noisy pixels or boundary artifacts during the α and β calibration. A parameter sweep was conducted across four preprocessing choices: **winsorization**, **erosion**, **minimum ground-truth standard deviation**, and **fit space**.

Winsorization removes outliers from each frame by clipping the lower and upper extremes of the depth prediction. I tested symmetric percentile ranges: [(0, 100), (1, 99), (2, 98), (3, 97), (5, 95), (10, 90)].

Erosion removes pixels from the edge of the mask. I tested erosion radii of [0, 1, 2, 3, 5, 10] pixels. This was done because mask boundaries are where the model can blend tree predictions with background predictions.

Minimum GT standard deviation removes frames from the α and β calibration if the masked trunk has too little depth variation. Flat or nearly flat regions can skew the calibration fit. I tested thresholds of [0, 0.02, 0.05, 0.10, 0.25] meters.

Fit space compares fitting α and β in depth space versus inverse-depth space. In depth space, the fit is: $D_{\text{gt}} = \alpha D_{\text{pred}} + \beta$ In inverse-depth space, the fit is: $\frac{1}{D_{\text{gt}}} = \alpha \frac{1}{D_{\text{pred}}} + \beta$ Inverse-depth space was evaluated because depth models can sometimes exhibit a more linear relationship to disparity-like values, and inverse-depth fitting can reduce the influence of errors at larger distances.

The goal of the sweep was to find the best combination of these four parameters. Each configuration was evaluated using an 80/20 train-validation split, and the best configuration was the one that minimized validation RMSE.

2.4. Anchor Box Calibration

Trunk-only calibration is limited because it relies on ground-truth depth maps, which are usually unavailable during real-world deployment. To address this, I introduced

a fixed object in the image as a practical metric-scale reference. I refer to this object as a **box anchor**. This setup is meant to match a possible deployment case where part of the robot arm or a known tool is visible in the camera view. Since the box anchor was placed at a known distance from the camera, its pixels provided an additional source of metric depth information.

I compared calibration using only trunk pixels with calibration that also used box pixels to estimate the linear calibration parameters α and β. The general weighted least-squares objective was:

$$\min_{\alpha,\beta}; \sum_{i\in\text{trunk}} (\alpha x_i+\beta-y_i)^2+\lambda \sum_{j\in\text{box}} (\alpha x_j+\beta-y_{\text{box},j})^2. \quad (2)$$

In this equation, x_i is the monocular model prediction, y_i is the Blender ground-truth depth, and λ determines how much the box pixels influence the fit. Because the box is at a known distance from the camera, it can act as a scale reference when trunk ground truth is not available. I evaluated four box-anchor calibration strategies.

The first strategy was **fixing β from the box**. For this setting, I fit α using trunk pixels after centering with the box mean. Then I calculated β from the box mean, this tested whether the box could provide a stable offset while the trunk pixels determined the scale.

The second strategy was **fixing α from the box**. For this setting, I set α as the ratio between the mean box ground-truth depth and the mean box prediction. Then I estimated β as the average residual over the trunk pixels. This tested whether the known box depth could provide a useful scale factor.

The third strategy was **per-shot α with a global β**. Since the camera shots came from repeated camera settings, I fit a separate α for each shot while keeping one shared β across all shots. This allowed each camera view to adjust its own scale while still using a single global offset.

The fourth strategy was a λ **merged sweep**. In this setting, I tested the weighted objective in Eq. 2 by sweeping the box weight over $\lambda \in \{0, 10^{-5}, 10^{-4}, 0.005, 0.05, 0.1, 0.25, 1.0, 2.0\}$. This tested whether a small amount of box supervision could improve calibration without dominating the trunk pixels.

This ablation tested whether a known object in the frame could replace, or at least reduce, the need for trunk ground-truth calibration. Trunk-only calibration is useful for analysis, but it is not practical for deployment unless a known metric reference is available in a scene. Per-strategy box-anchor results for both datasets are reported in Tabs. 3 and 10.

2.5. Fine-Tuning Depth Anything V2

After testing zero-shot monocular depth models and calibration, I fine-tuned Depth Anything V2 to see whether synthetic supervision from Blender could produce a stronger metric depth signal. I followed the official Depth Anything V2 metric depth training setup as closely as possible, but used my synthetic orchard dataset instead of the original training datasets [11]. The input was an RGB frame, and the supervision came from the Blender ground-truth depth map. Training used the Scale-Invariant Log loss, or SiLog loss, on valid masked pixels:

$$\mathcal{L}_{\text{SiLog}} = \sqrt{\frac{1}{n}\sum_i d_i^2 - \lambda_{\text{silog}} \left(\frac{1}{n}\sum_i d_i\right)^2}, \quad d_i = \log y_i - \log \hat{y}_i. \quad (3)$$

Here, $\hat{y}_i$ is the predicted depth, y_i is the ground-truth depth, and the loss is computed only on the selected mask pixels. I mainly used trunk-masked supervision because the trunk is the most stable pruning-relevant structure. This also kept background pixels from dominating the loss. The model was evaluated using validation RMSE in meters.

The first fine-tuning setup used only trunk pixels in the loss. This tested whether Depth Anything V2 could learn the metric scale of the synthetic orchard domain from the trunk region alone. It also gave the most direct comparison to the calibration experiments because both focused on improving metric depth on tree pixels.

I also tested whether adding the fixed box object during training could improve metric reasoning. The idea was similar to the anchor calibration experiment: if a known object appears in the frame, the model may learn a stronger scale cue. To test this, I trained with different loss-mask settings using the trunk mask and box mask. ju j u7j u7mj uk The first loss-mask setting was the **union loss**. In this setting, the loss was computed over the union of trunk pixels and box pixels: $\mathcal{L}_A = \mathcal{L}_{\text{SiLog}}(T \cup B)$. This tested whether simply adding box pixels to the supervision helped.

The second setting was the **weighted loss**. In this setting, the trunk and box pixels were combined, but the box pixels were reweighted so that differences in mask area did not dominate the loss: $\mathcal{L}_B = \mathcal{L}_{\text{SiLog}}(T \cup wB)$. This tested whether balancing the pixel contribution improved the training signal.

The third setting was the **balanced loss**. In this setting, the trunk and box losses were computed separately and then averaged: $\mathcal{L}_C = 0.5\,\mathcal{L}_{\text{SiLog}}(T) + 0.5\,\mathcal{L}_{\text{SiLog}}(B)$. This forced the model to treat trunk and box supervision as separate depth signals.

The fourth setting was the **anchor loss**. For this setting, I used trunk SiLog loss and added a box-only anchor term: $\mathcal{L}_D = \mathcal{L}_{\text{SiLog}}(T) + \lambda_{\text{box}}\mathcal{L}_{\text{box}}(B)$. The trunk still provided the main dense depth supervision, while the box acted as an

additional scale anchor.

Here, T represents the trunk mask pixels, B represents the box mask pixels, and wB represents the box pixels after applying a weighting term so that the box does not dominate only because of the number of pixels. The term $\mathcal{L}_{\text{SiLog}}$ is the scale-invariant log loss used for the depth prediction, while $\mathcal{L}_{\text{box}}(B)$ is the box-only anchor loss. In my final anchor setting, I used $\lambda_{\text{box}} = 0.2$, so the box helped guide the absolute scale without becoming the main training signal. Full per-loss results are tabulated in Full per-loss results are tabulated in Tabs. 6 and 12. .

I also used curriculum sampling during training. At first, the model sampled from more controlled camera views. After a few epochs, I added more randomized camera poses to the training set. This let the model learn the basic trunk-depth relationship before handling more pose variation. The randomized views made the training data more realistic, but they also made the task harder.

These fine-tuning experiments tested whether metric depth could be improved by giving the monocular model direct synthetic supervision instead of only applying calibration after prediction. This became a key part of the project because the fine-tuned Depth Anything V2 model produced the strongest single-view results.

2.6. Multi-View Depth Refinement

The final set of experiments investigated whether incorporating stereo and multi-view inputs could improve the accuracy of monocular depth predictions. The aim was not to perform classical stereo matching, but rather to train neural refinement models that receive both RGB images and initial monocular depth predictions as input, and produce a refined metric depth map as output. This approach was motivated by the challenges posed by dormant tree scenes, where branches are thin, sparse, and frequently overlap with the background, making depth estimation from a single image particularly difficult.

Each training sample consisted of multiple spatially adjacent camera views. Experiments were conducted using either one stereo pair or three stereo pairs, corresponding to two or six views, respectively. The number of views provided during training was controlled via the multi-view loader implemented in the codebase. For each view, the dataset included an RGB image, an input depth map, a ground-truth Blender depth map, a mask, camera intrinsics, and camera pose information. The source of the input depth map varied by experiment, and included outputs from PRO, calibrated PRO, or the fine-tuned Depth Anything V2 model.

The loss function used for training these models was consistent with the single-view fine-tuning setup. Specifically, models were optimized using SiLog loss computed over valid masked pixels, and performance was evaluated using masked RMSE measured in meters. This ensured that results from the multi-view experiments could be directly compared to those from single-view Depth Anything V2 fine-tuning.

2.6.1 Shared CNN Stereo U-Net

The first refinement model employed was a shared CNN stereo U-Net, implemented in the code as MVStereoUNet. This model follows an encoder-decoder architecture and is trained from scratch. Each camera view is processed by the same encoder, resulting in shared encoder weights across all views. This design choice is significant because it enables the model to learn a unified feature extractor applicable to any view, rather than maintaining separate parameters for each camera.

In the RGB+Depth configuration, the RGB image and input depth map are concatenated prior to being input to the encoder. This allows the encoder to access both appearance information from the RGB channels and metric information from the monocular depth prediction. A depth-only ablation was also conducted, in which the RGB channels were omitted and the model received only the input depth map. This setting is controlled in the code by the no_rgb flag, which switches the encoder input from RGB+D to depth-only.

A key aspect of the model architecture is the fusion step. In this context, fusion refers to the combination of features at the bottleneck level within the network, rather than blending images or averaging depth maps. Each view is first encoded independently to produce its own bottleneck feature map. These bottleneck features from all views are then concatenated along the channel dimension. Subsequently, a learned 1×1 convolution is applied to mix the concatenated channels and reduce them back to the standard bottleneck size. This mechanism enables the model to integrate information from multiple views before the decoder predicts the refined depth map.

The fusion ablation is critical for determining whether cross-view feature exchange contributes to improved performance. A shared encoder alone does not guarantee that information is exchanged between views. When fusion is disabled, each view is processed with shared encoder weights but decoded independently using only its own bottleneck features. When fusion is enabled, the decoder receives a bottleneck representation that integrates information from all views. Thus, the fusion on/off ablation specifically tests the impact of cross-view feature exchange, rather than solely the effect of shared encoder weights.

The decoder produces a refined depth map for each view. Skip connections are maintained on a per-view basis, allowing the decoder to utilize local spatial information specific to each view. A positive activation function is applied to the final depth output to ensure that predicted depth val-

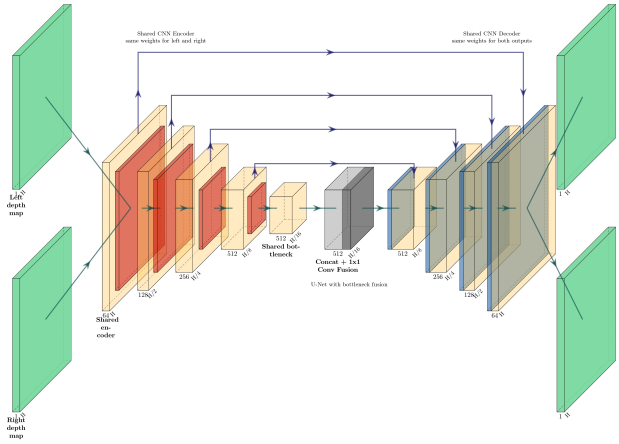

Figure 4. Shared CNN stereo U-Net used for multi-view depth refinement. Each view is encoded with shared weights. Fusion occurs at the bottleneck by concatenating view features and applying a learned 1×1 convolution before decoding refined metric depth.

ues remain valid. This CNN model served as the baseline for evaluating whether incorporating stereo or multi-view context can improve the quality of noisy monocular depth maps.

2.6.2 DINOv2 RGB-D Refinement Model

The second refinement model utilized a frozen DINOv2 ViT-L RGB encoder in conjunction with a trainable depth-side branch, implemented in the code as MVStereoDINOUNet. This model adopts the same general multi-view refinement strategy as the CNN model, but substitutes the from-scratch RGB feature extraction with pretrained DINOv2 features.

The DINO-based model processes RGB and depth inputs differently from the CNN model. Whereas the CNN model concatenates RGB and depth channels at the input, the DINO model employs separate processing branches. The RGB image is passed through the frozen DINOv2 ViT-L encoder, while the input depth map is processed by a trainable CNN depth-side branch. This separation is necessary because DINOv2 is an RGB vision transformer and does not accept a depth channel as input.

The DINOv2 backbone supplies pretrained RGB features that can capture branch boundaries, texture, and overall scene structure. The depth-side branch learns features from the input depth map, which provides the metric depth signal from either PRO or fine-tuned Depth Anything V2. These two feature streams are combined via trainable projection and fusion layers before being input to the decoder. In the implementation, the DINOv2 backbone remains largely frozen, while the depth-side branch, projection layers, fusion layers, and decoder are trainable.

The cross-view fusion step in the DINO model is conceptually identical to that of the CNN model. Each view produces its own RGB-D bottleneck feature, and these bottleneck features from all views are concatenated along the channel dimension. A learned 1×1 convolution then mixes the information from the different views into a single fused representation, which is subsequently used by the decoder to predict a refined depth estimate. If fusion is disabled, the model decodes each view independently without direct exchange of bottleneck information across views.

Several ablation studies were conducted to determine the contribution of different model components. The RGB+Depth setting utilized both RGB features and the input depth map. The RGB-only setting excluded the depth signal to assess whether pretrained DINO features alone were sufficient. The depth-only setting removed the RGB pathway to evaluate how much information could be learned from the depth map alone. The fusion-off setting tested the effect of disabling cross-view feature exchange. Additionally, experiments compared the use of one stereo pair versus three stereo pairs to assess whether increasing the number of views improved the final RMSE.

The best performance for the DINO refinement model was achieved when the input depth map was generated by the fine-tuned Depth Anything V2 model. This indicates that the pretrained RGB encoder provided the greatest benefit when the input depth map already contained a strong metric signal. In cases where the input depth was noisy or poorly scaled, the refinement model had less reliable information to correct, resulting in diminished performance. Encoder comparisons, calibration, and fusion ablation results for both datasets are in Tabs. 7, 8, 13 and 14.

3. Data Diversity Ablation

Using synthetic data made it possible to generate a large training set through augmentation. The initial dataset contained 42k+ frames for fine-tuning Depth Anything V2. Training on the entire dataset was computationally expensive, requiring about two days on the HPC DGXH node with early stopping based on validation accuracy. This limited the ability to iterate quickly, which was a challenge when analyzing how to compose the training data.

To make experiments more manageable, I switched to a smaller ablation setup using 10 trees. The main questions were whether trunk masking was important and how the composition of training frames affected validation accuracy. I sampled frames from 10 trees with the bark_brown_02 texture and used an 80/20 split at the tree level, assigning 8 trees for training and 2 for validation.

I evaluated three data configurations, keeping the number of frames fixed at 432 for training and 108 for validation. The first configuration, half-and-half, used 4 regular camera poses and 4 fixed box poses. The second, box-only, used 8 fixed box poses. The third, cam-only, used 8 regular camera poses. Each configuration was tested both with and without trunk masking. I repeated the experiments across 5

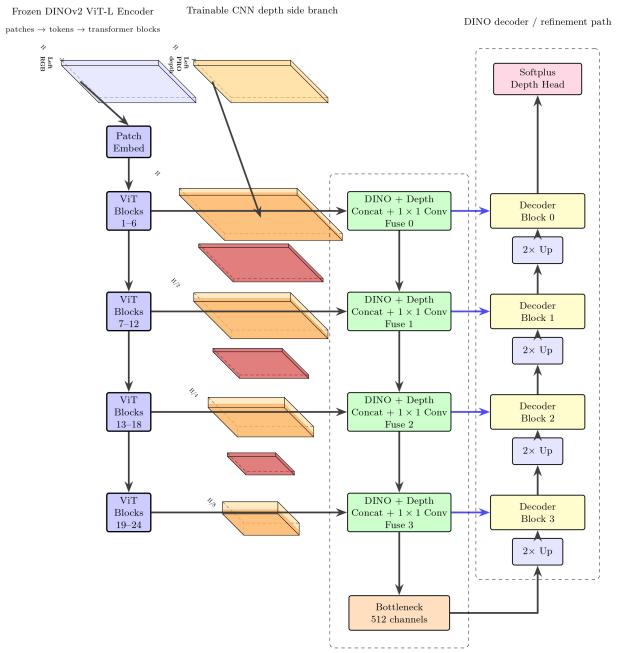

Figure 5. DINOv2 RGB-D refinement model. RGB images are encoded with a frozen DINOv2 ViT-L backbone, while input depth maps are encoded by a trainable depth-side branch. RGB and depth features are fused before the decoder predicts refined metric depth.

Table 1. Main depth estimation results. RMSE is reported in meters.

Method	RMSE	Notes
DA2-ft single-view	0.0550	Full-tree depth
CNN RGB+D refinement	0.0503 ± 0.0090	DA2-ft depth, 4 pairs
DINO RGB+D refinement	$\mathbf{0.0445 \pm 0.0057}$	DA2-ft depth, 3 pairs
SPUR anchor loss	0.1136 ± 0.0097	Box as scale anchor

random seeds by varying which trees were sampled, resulting in 30 experiments trained for up to 80 epochs.

The results from these ablations informed the later fine-tuning setup. Masking was important because tree pixels are only a small fraction of the image. The data composition tests also clarified how to include regular camera views and box views in subsequent experiments. Full Per-configuration numbers for the refinement studies are reported in Tabs. 13 to 15 for SPUR and Tabs. 7 and 8 for TRUNK, with the best per-encoder TRUNK results in Tab. 16.

4. Results

Fine-tuning Depth Anything V2 on synthetic orchard data produced the strongest single-view result. Synthetic supervision provided a stronger metric signal than calibrating zero-shot monocular depth outputs. The best full-tree result was achieved by the DINO RGB+D refinement model using DA2-ft depth and three stereo pairs, reaching 0.0445 m RMSE. This indicates that multi-view refinement was most effective when the input depth map was already accurate.

4.1. Ablation Summary

The ablations showed that the depth source mattered more than the architecture alone. When the refinement models used Raw PRO depth, RMSE stayed higher. When the input depth came from the fine-tuned DA2 model, both CNN and DINO improved, meaning the refinement model worked best when the monocular depth already had a reliable metric signal. The DINO model performed best with DA2-ft depth because the frozen DINOv2 encoder provided strong RGB features, and the trainable depth branch preserved the input depth signal. The CNN stereo U-Net served as a simpler baseline for testing RGB+D input, fusion, and the effect of view count.

The fusion ablation showed that fusion improved results more for CNN than for DINO. In this context, fusion refers to bottleneck feature fusion, where view features are concatenated and mixed with a learned 1x1 convolution before decoding. Without fusion, each view is decoded independently. These results showed that fusion alone could not compensate for a weak input depth source.Increasing the number of views did not always improve performance. DINO achieved the best results with three stereo pairs, while CNN performed best with four stereo pairs. The SPUR ablation showed that the box was most effective as an auxiliary scale anchor rather than being merged into the training mask. Per-configuration numbers for the SPUR refinement studies are reported in Tabs. 13 to 15.

5. Conclusion

We address metric depth estimation for robotic pruning in orchard environments, focusing on dormant apple trees. The main goal is to determine whether calibration, synthetic fine-tuning, or stereo and multi-view refinement yields the most accurate pruning depth estimates. Because collecting dense real-world depth data in orchards is difficult, we generated synthetic RGB images, ground-truth depth maps, masks, and camera poses using Blender.

Experimental results show that fine-tuning Depth Anything V2 on synthetic orchard data leads to the most accurate single-view depth predictions. Zero-shot models and calibration served as baseline comparisons, but neither alone was sufficient for effective performance. While calibration addressed some scale errors, the most significant gains came when the model was trained directly with synthetic metric depth supervision.

Refinement models using Raw PRO depth showed limited improvement, while models using fine-tuned DA2

depth performed better in both CNN- and DINO-based refinement settings. The best result was with the DINO RGB+D model, which used DA2-ft depth and three stereo pairs, resulting in an RMSE of 0.0445 m. These results indicate that pretrained RGB features and multi-view information are most effective when the initial depth input is reliable.

Ablation experiments showed that both masking and the box-anchor signal contributed to performance, but the dominant trend remained: improving the input depth signal via synthetic fine-tuning was the primary factor influencing overall accuracy.

The next work will involve sim-to-real evaluation on orchard data to assess generalization. Runtime considerations are critical for deployment. The DINO RGB+D model processed a 6-view group in about 484.7 ms, or 80 ms per view. Deploying a single stereo pair on a robotic platform would reduce views to two, potentially enabling near-real-time operation, though this requires testing on actual hardware. Further research should investigate direct fine-tuning of stereo or multi-view depth models rather than restricting their use to refinement after monocular prediction.

References

[1] Shariq Farooq Bhat, Reiner Birkl, Diana Wofk, Peter Wonka, and Matthias Müller. Zoedepth: Zero-shot transfer by combining relative and metric depth. *arXiv preprint arXiv:2302.12288*, 2023. 2, 3

[2] Blender Foundation. *Blender Manual: Render Passes*, 2026. Official documentation. 1

[3] Abhinav Jain, Cindy Grimm, and Stefan Lee. Learning to prune branches in modern tree-fruit orchards, 07 2025. 1

[4] Byeongjun Kwon and Munchurl Kim. One look is enough: A novel seamless patchwise refinement for zero-shot monocular depth estimation models on high-resolution images. *arXiv preprint arXiv:2503.22351*, 2025. 1, 2

[5] Haotong Lin, Sili Chen, Junhao Liew, Donny Y. Chen, Zhenyu Li, Guang Shi, Jiashi Feng, and Bingyi Kang. Depth anything 3: Recovering the visual space from any views. *arXiv preprint arXiv:2511.10647*, 2025. 1, 2

[6] Alessandro Navone, Mauro Martini, and Marcello Chiaberge. Autonomous robotic pruning in orchards and vineyards: a review. *arXiv preprint arXiv:2505.07318*, 2025. 1, 2

[7] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023. 2

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, 2015. 2

[9] Bill F. Smith. Tilingzoedepth. https://github.com/BillFSmith/TilingZoeDepth, 2026. GitHub repository, accessed March 23, 2026. 3

[10] Tieqiao Wang, Abhinav Jain, Liqiang He, Cindy Grimm, and Sinisa Todorovic. A dataset for semantic and instance segmentation of modern fruit orchards. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*, pages 5420–5430, June 2025. 1, 2

[11] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. *arXiv preprint arXiv:2406.09414*, 2024. 1, 2, 5

[12] Alexander You, Cindy M. Grimm, Abhisesh Silwal, and Joseph R. Davidson. Semantics-guided skeletonization of upright fruiting offshoot trees for robotic pruning. *Computers and Electronics in Agriculture*, 192:106622, 2022. 2

[13] Zhichao Zheng, Henry Williams, and Bruce A. MacDonald. Orcharddepth: Precise metric depth estimation of orchard scene from monocular camera images. *arXiv preprint arXiv:2502.14279*, 2025. 1, 2

A. Code Repository

A cleaned GitHub repository with the training scripts, experiment setup, and supporting code is provided at:

https://github.com/joses2017smjh/spur-da2ft-depth-e

B. Additional Experimental Results

All ablation studies share the same protocol: bark_brown_02, 10 trees per seed (8 train + 2 val) under an 80/20 tree-level split, 5 random seeds, 80 epochs with patience-10 early stopping, no pose, and the nearest-GT loader. Trunk and SPUR experiments differ only in the source captures: *trunk* uses full_trunk, *SPUR* uses the reorganized full_spur box-family captures.

B.1. Trunk Experiments

Table 2. TRUNK — Validation RMSE by model and processing stage (full_trunk, box_cam stereo). Mean ± std over 5 seeds.

Model	Uncal. RMSE (m)	Calib. RMSE (m)	Best preproc. (m)
DA3 Metric	25.714 ± 1.060	0.367 ± 0.029	0.178 ± 0.068
DA3 Relative	1.701 ± 0.109	0.372 ± 0.032	0.191 ± 0.083
PRO	1.455 ± 0.253	$\mathbf{0.279 \pm 0.034}$	$\mathbf{0.136 \pm 0.023}$

Table 3. TRUNK — Box-anchor strategies on each model's Phase-1 best configuration. RMSE in meters.

Model	Base	Fix β	Fix α	λ-merge
DA3 Metric	0.181 ± 0.065	0.961 ± 0.148	0.178 ± 0.050	$\mathbf{0.177 \pm 0.068}$
DA3 Relative	0.200 ± 0.094	1.288 ± 0.412	0.244 ± 0.092	$\mathbf{0.195 \pm 0.097}$
PRO	0.143 ± 0.016	0.828 ± 0.492	0.271 ± 0.146	$\mathbf{0.138 \pm 0.008}$

Table 4. TRUNK — View-diversity comparison. Val RMSE on the trunk mask.

Data config	Val RMSE (m)
Half-and-half	0.182 ± 0.072
Box-only	$\mathbf{0.166 \pm 0.046}$
Cam-only	0.177 ± 0.041

Table 5. TRUNK — Data composition ablation on fine-tuning. Mean $\pm$ std over 5 seeds, 80 epochs.

Data config	Masking	Val RMSE (m)	Best epoch
Half-and-half	Trunk mask	0.182 ± 0.072	54 ± 28
Box-only	Trunk mask	$\mathbf{0.166 \pm 0.046}$	54 ± 32
Cam-only	Trunk mask	0.177 ± 0.041	40 ± 26
Half-and-half	No mask	0.437 ± 0.006	76 ± 2
Box-only	No mask	0.449 ± 0.004	77 ± 3
Cam-only	No mask	0.448 ± 0.009	76 ± 2

Table 6. TRUNK — Box-anchor loss ablation on fine-tuning. Mean $\pm$ std over 5 seeds.

Mode	**Loss / Box Role**	**RMSE (m)**
A union	$\mathrm{SiLog}(T \cup B)$; weak signal	0.1838 ± 0.0093
B weighted	$\mathrm{SiLog}(T \cup wB)$; equalized signal	0.1558 ± 0.0116
C balanced	$0.5\,\mathrm{SiLog}(T) + 0.5\,\mathrm{SiLog}(B)$; split objective	0.1363 ± 0.0119
D anchor	$\mathrm{SiLog}(T) + 0.2L_1(B)$; scale anchor	$\mathbf{0.1136 \pm 0.0097}$

Table 7. TRUNK — RGB / depth / pretraining comparison. 1 stereo pair, fusion on.

Encoder	Inputs	Depth source	Val RMSE (m)
DINOv2 (frozen)	RGB only	—	0.0859 ± 0.0107
DINOv2 (frozen)	RGB + D sidebr.	Raw PRO	0.0822 ± 0.0148
DINOv2 (frozen)	RGB + D sidebr.	DA2-ft	$\mathbf{0.0399 \pm 0.0093}$
CNN (scratch)	D only	Raw PRO	0.0534 ± 0.0105
CNN (scratch)	RGB + D	Raw PRO	0.0440 ± 0.0069
CNN (scratch)	RGB + D	DA2-ft	0.0450 ± 0.0103

Table 8. TRUNK — Calibration / fusion ablation. 1 stereo pair, RGB + D.

Enc.	Calib	Fus.	Depth	Val RMSE (m)	Best epoch
DINO	raw PRO	on	pro	0.0822 ± 0.0148	23.8 ± 7.0
DINO	raw PRO	off	pro	0.0864 ± 0.0156	21.2 ± 15.3
DINO	α, β	on	pro	0.0826 ± 0.0102	29.0 ± 13.7
DINO	α, β	off	pro	0.0842 ± 0.0082	26.6 ± 6.5
DINO	—	off	DA2-ft	0.0406 ± 0.0107	20.0 ± 12.4
CNN	raw PRO	on	pro	0.0440 ± 0.0069	30.4 ± 12.9
CNN	raw PRO	off	pro	0.0551 ± 0.0108	35.6 ± 20.4
CNN	α, β	on	pro	0.0449 ± 0.0105	31.4 ± 13.6
CNN	α, β	off	pro	0.0513 ± 0.0085	28.4 ± 8.1
CNN	—	off	DA2-ft	$\mathbf{0.0420 \pm 0.0102}$	23.4 ± 14.4

B.2. SPUR Experiments

Table 9. SPUR — Validation RMSE by model and processing stage (`full_spur`, box_cam stereo). Mean ± std over 5 seeds.

Model	Uncal. RMSE (m)	Calib. RMSE (m)	Best preproc. (m)
DA3 Metric	18.990 ± 0.900	0.534 ± 0.027	$\mathbf{0.138 \pm 0.050}$
DA3 Relative	1.706 ± 0.059	0.537 ± 0.027	0.237 ± 0.088
PRO	1.801 ± 0.130	$\mathbf{0.421 \pm 0.005}$	0.202 ± 0.084

Table 10. SPUR — Box-anchor strategies on each model's Phase-1 best configuration. RMSE in meters.

Model	Base	Fix β	Fix α	λ-merge
DA3 Metric	0.189 ± 0.127	0.625 ± 0.454	0.397 ± 0.382	$\mathbf{0.187 \pm 0.128}$
DA3 Relative	0.249 ± 0.063	0.552 ± 0.251	$\mathbf{0.234 \pm 0.085}$	0.242 ± 0.067
PRO	0.207 ± 0.074	0.615 ± 0.288	0.368 ± 0.086	$\mathbf{0.203 \pm 0.077}$

Table 11. SPUR — View-diversity comparison. Val RMSE on the SPUR mask.

Data config	Val RMSE (m)
Half-and-half	0.1155 ± 0.0156
Box-only	$\mathbf{0.0972 \pm 0.0090}$
Cam-only	0.1009 ± 0.0097

Table 12. SPUR — Box-anchor loss ablation on fine-tuning. Mean ± std over 5 seeds.

Mode	**Loss / Box Role**	**RMSE (m)**
A union	$\mathrm{SiLog}(T \cup B)$; weak signal	0.1838 ± 0.0093
B weighted	$\mathrm{SiLog}(T \cup wB)$; equalized signal	0.1558 ± 0.0116
C balanced	$0.5\,\mathrm{SiLog}(T) + 0.5\,\mathrm{SiLog}(B)$; split objective	0.1363 ± 0.0119
D anchor	$\mathrm{SiLog}(T) + 0.2L_1(B)$; scale anchor	$\mathbf{0.1136 \pm 0.0097}$

Table 13. SPUR — RGB / depth / pretraining comparison. 1 stereo pair, fusion on.

Encoder	Inputs	Depth source	Val RMSE (m)
DINOv2 (frozen)	RGB only	—	0.143 ± 0.013
DINOv2 (frozen)	RGB + D sidebr.	Raw PRO	0.139 ± 0.014
DINOv2 (frozen)	RGB + D sidebr.	DA2-ft	$\mathbf{0.046 \pm 0.006}$
CNN (scratch)	D only	Raw PRO	0.118 ± 0.013
CNN (scratch)	RGB + D	Raw PRO	0.105 ± 0.009
CNN (scratch)	RGB + D	DA2-ft	0.056 ± 0.006

Table 14. SPUR — Calibration / fusion ablation. 1 stereo pair, RGB + D.

Enc.	Calib	Fus.	Depth	Val RMSE (m)	Best epoch
DINO	raw PRO	on	pro	0.139 ± 0.014	37.2 ± 15.9
DINO	raw PRO	off	pro	0.139 ± 0.012	30.2 ± 13.7
DINO	α, β	on	pro	0.139 ± 0.013	29.4 ± 8.9
DINO	α, β	off	pro	0.138 ± 0.010	30.0 ± 7.1
DINO	—	off	DA2-ft	0.048 ± 0.007	25.4 ± 14.8
CNN	raw PRO	on	pro	0.105 ± 0.009	51.2 ± 21.4
CNN	raw PRO	off	pro	0.111 ± 0.014	45.4 ± 19.4
CNN	α, β	on	pro	$\mathbf{0.100 \pm 0.007}$	51.8 ± 6.5
CNN	α, β	off	pro	0.106 ± 0.005	48.4 ± 20.6
CNN	—	off	DA2-ft	0.053 ± 0.002	28.2 ± 15.3

B.3. Best Configurations

Table 15. SPUR — Stereo-pair sweep. RGB + D, DA2-ft depth, fusion on.

		DINO		CNN	
# pairs	n_v	RMSE (m)	Best ep.	RMSE (m)	Best ep.
1	2	0.0461 ± 0.0058	30.4 ± 10.3	0.0555 ± 0.0062	22.0 ± 10.6
2	4	0.0461 ± 0.0055	26.0 ± 10.1	0.0551 ± 0.0051	16.6 ± 4.8
3	6	$\mathbf{0.0445 \pm 0.0057}$	22.8 ± 4.3	0.0597 ± 0.0145	14.8 ± 8.6
4	8	0.0453 ± 0.0065	25.0 ± 10.3	$\mathbf{0.0503 \pm 0.0090}$	33.4 ± 19.6

Table 16. TRUNK — Two best configurations (per encoder). RGB + D, DA2-ft depth, 1 stereo pair.

Encoder	Depth	Fusion	Val RMSE (m)	Best epoch
DINOv2 (frozen)	DA2-ft	on	$\mathbf{0.0399 \pm 0.0093}$	19.8 ± 5.8
CNN (scratch)	DA2-ft	off	0.0420 ± 0.0102	23.4 ± 14.4