Explicit model pre-training for 3D occupancy prediction

Pontus Brandin,
Albert Henrik Mattsson

Master's thesis
2026:E32

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Abstract

Understanding 3D scene structure from camera input is a central challenge in autonomous driving. Recent methods often rely on vision foundation models or annotated occupancy labels to provide strong semantic and geometric supervision, but these dependencies can limit scalability and introduce external biases. This thesis investigates whether an explicit 3D Gaussian scene representation can be pre-trained in a more self-contained manner from multi-view camera data and optional LiDAR supervision. Building on GaussTR [16], we replace foundation-model feature inputs with a ResNet backbone and replace depth supervision from Metric3D [13] with self-supervised geometric consistency losses, optionally supported by LiDAR depth supervision. The learned representation is evaluated through binary occupancy prediction on Occ3D-nuScenes [22]. Our results show that meaningful explicit 3D representations can be learned without vision foundation models at inference, and that fully self-supervised pre-training remains viable, although with reduced occupancy performance compared to VFM-assisted baselines. When adapted for semantic occupancy prediction, the proposed model achieves competitive results, including stronger mIoU than the original GaussTR baseline in our comparison. The findings suggest that geometric understanding and feature separation are closely linked. Improved geometry facilitates semantic learning, while stronger features support more accurate spatial representations.

Acknowledgements

We would like to express our sincere gratitude to our supervisor at Zenseact, Amer Mustajbasic. His deep domain knowledge, continuous guidance, and dedicated support were invaluable throughout this thesis. Our daily discussions with him played a crucial role in driving the project forward, and his ability to provide feedback and creative ideas for future directions greatly contributed to the quality of this work.

We would also like to extend our appreciation to our academic supervisors at LTH, Viktor Larsson and Johanna Lidholm. Their thoughtful feedback, encouragement, and ability to view the project from different perspectives were of great help to us.

Contents

1 Introduction

Understanding the three-dimensional structure of a scene from visual input is a fundamental problem in computer vision. One major area which applies this field is autonomous driving (AD). In AD tasks, it is essential that the system has a clear representation of the vehicle's surroundings. Our work aims to pre-train a vision-based model which produces an internal representation of the surroundings which supports multiple different downstream tasks, for instance object detection or path planning.

1.1 Implicit and Explicit Representations

A key distinction in world models is the one between implicit and explicit representations. Implicit models encode the scene in weights in a neural network with occupancy, density or some other quantity as output. This allows for efficient and continuous representations, but it makes the scene structure difficult to interpret and control. In contrast, explicit methods model a scene with a discrete set of elements with parameters which have direct physical interpretations. These representations are easier to interpret and manipulate, which can be beneficial for tasks which require geometric reasoning.

One family of explicit models are voxel-based. They discretize 3D space into voxels, assigning properties to each voxel. This approach can suffer from being to computationally expensive, as it inevitably uses much computing power to model empty space. Another option is to use a bird's eye view (BEV) model, which flattens the height dimension, modelling the scene in 2D, as seen from above. This is more efficient than 3D voxels, but has the obvious drawback of losing information along the compressed dimension. Instead of these approaches, we take inspiration from recent works which use a set of Gaussians to model the scene. This has a few advantages. Gaussians can cluster in space at points which are actually occupied, making better use of computational resources than voxel methods, while still maintaining three dimensional spatial information. Additionally, Gaussians are well suited to model different shapes, since their positions and covariances are versatile and can be adjusted.

1.2 Supervision

As with any deep learning task, training 3D perception models requires supervision signals that guide the model toward learning meaningful representations. Different types of supervision vary in how costly they are to obtain, how scalable they are, and what kind of information they provide.

One common approach is to use annotations, such as semantic occupancy grids, where semantic means that different object categories are separated and labelled according to their class. Annotations provide strong and direct supervision, but they are expensive and time-consuming to produce. This is especially true in 3D settings, where labelling requires detailed understanding of the spatial structure of the scene. In practice, such labels are often not created fully manually, but instead generated using other models and sensor data, for example by combining LiDAR measurements with annotation pipelines. One example of a work which has generated voxelized 3D semantic occupancy annotations this way is Occ3D [22], which we use for validation. While this reduces the manual effort, it introduces additional complexity and dependency on upstream models, and may propagate their errors and biases into the training data.

Vision foundation models (VFMs) are large-scale models trained on big amounts of image data to learn representations of an image. These models can be used as a source of supervision by extracting feature representations from input images. Instead of relying on manually annotated data, a model can be trained to match or reconstruct these features, effectively transferring knowledge from the VFM. In some literature, [16] for example, this is referred to as a self-supervised approach, since it eliminates the need for annotations. However, since it introduces a dependency on external pre-trained models and their learned biases, we will not consider it to be full self-supervision and aim to avoid VFM dependence.

LiDAR sensors provide accurate geometric measurements of the environment and are often used to generate ground truth for 3D tasks. This type of supervision is widely used because of its precision. However, it requires additional hardware and restricts training to datasets where LiDAR data is available. Since LiDAR is more expensive than cameras, it can be hard to acquire datasets with LiDAR data. Additionally, LiDAR can only detect the surface of objects and not full three-dimensional geometry, so it cannot generate perfect occupancy ground truth.

Another option is to rely entirely on camera images, where the training signal is derived from the input data itself. In multi-view settings, images of the same scene from different viewpoints provide natural geometric constraints. These can be used to enforce consistency between views and guide the learning process without external supervision. This has the big advantage that any raw multi-view camera or video data can be used, making training data much more easily accessible.

1.3 Pre-Training

Many tasks in autonomous driving require an understanding of the surrounding scene. These tasks can be handled by separate models with different architectures and training data. This divison leads to duplicated effort and a reliance on large amounts of

task-specific data. It is therefore desirable to learn one representation of the environment that can support multiple downstream tasks. This motivates the use of pre-training to learn a general-purpose 3D representation, which can later be reused or fine-tuned for different applications. In this work, we focus on pre-training a model that learns a general 3D representation of the scene rather than optimizing for a single task. This representation can then be transferred to multiple downstream tasks, where it can be fine-tuned, potentially with task-specific supervision.

1.4 Occupancy Prediction

Occupancy prediction is the task of estimating whether regions in 3D space are occupied or free, represented as a discretized voxel grid. Semantic occupancy prediction refers to the case where each occupied voxel is also assigned a class label. This provides a representation of the scene which captures geometry and optionally semantics. In this work, occupancy prediction is used as a proxy metric for evaluating general 3D spatial understanding. While it is itself a specific task, it requires the model to reason about geometry, depth, and spatial consistency, making it a useful indicator of the quality of the learned representation. Importantly, the goal of the proposed method is not to optimize specifically for occupancy prediction, but to learn a general-purpose 3D representation that can support multiple downstream tasks. Therefore, during validation, we evaluate occupancy prediction performance without task-specific fine-tuning. Such fine-tuning could potentially be applied later to improve performance on this or other tasks.

1.5 Our Contribution and Research Questions

In this work, we investigate whether explicit 3D scene representations can be learned without relying on external supervision from VFMs or annotated data. Building on recent Gaussian-based approaches, we aim to develop a self-contained pretraining framework. The central idea is to replace external supervisory signals with self-supervised objectives derived from geometric and photometric consistency, to the extent in which it is possible while maintaining satisfactory performance. We distinguish between two types of VFM dependence: using them at inference or purely as a supervision signal. We explore whether a Gaussian representation can learn meaningful semantics without semantic VFM supervision or input, as well as if it can learn correct geometry and depth without depth supervision or a specialized depth VFM at inference. Additionally, we will explore the effect of using LiDAR as a supervision signal, even though our method is vision-based. As in [1], we will consider this a form of self-supervision since LiDAR is a form of raw data and not dependent on external training or label generation.

2 Theory

To provide the necessary context for the proposed method, this section presents the theoretical foundations underlying our approach. We introduce key concepts from machine learning, as well as geometric principles essential for multi-view 3D scene understanding.

2.1 Machine Learning Concepts

We first introduce some basic machine learning concepts. More comprehensive treatments can be found in standard machine learning textbooks.

2.1.1 Multilayer Perceptron

A multilayer perceptron (MLP) is a feedforward neural network that transforms an input feature vector into an output through a sequence of learned operations. It consists of an input layer, one or more hidden layers, and an output layer.

Each layer performs a linear transformation of its input followed by a nonlinear activation function. Given an input vector $\mathbf{x}$, the output of a layer can be written as

$$\mathbf{h} = \sigma(W\mathbf{x} + \mathbf{b})$$

where W is a learnable weight matrix, $\mathbf{b}$ is a learnable bias vector, and $\sigma(\cdot)$ denotes an activation function such as ReLU. The linear transformation combines information from the input features, while the nonlinear activation enables the network to model complex relationships that cannot be represented by linear mappings alone.

By stacking multiple layers, the network gradually transforms the input into increasingly useful feature representations. The parameters W and $\mathbf{b}$ are learned during training by minimizing a loss function using backpropagation.

In practice, MLPs are commonly used to map feature vectors from one representation to another. In the context of this work, MLPs are used to transform learned features into desired outputs, such as Gaussian parameters or semantic classes.

2.1.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are feedforward neural networks designed to process grid-structured data such as images. An image can be represented as a three-dimensional tensor of size $(H \times W \times C)$, where H and W denote the spatial dimensions and C the number of channels (e.g. RGB channels).

Unlike fully connected networks, CNNs exploit the spatial structure of the input by applying convolutional filters locally across the image. A convolutional layer consists

of a set of learnable kernels that are convolved with the input, producing feature maps that capture local patterns such as edges, textures, and shapes. By stacking multiple convolutional layers, the network can learn increasingly abstract representations of the input.

In addition to convolutional layers, CNNs often include pooling layers, which reduce the spatial resolution of feature maps while retaining important information. This helps decrease computational cost and increases robustness to small spatial variations.

2.1.3 Residual Networks

Residual networks (ResNets) [11] are highly relevant for this work, as they are commonly used as backbones for extracting features from image data. A ResNet is a convolutional neural network architecture that introduces so-called residual blocks with skip connections. These connections allow the input of a block to bypass one or more layers and be added to the output.

Instead of learning a direct mapping $H(x)$, the network learns a residual mapping $F(x)$ such that

$$y = F(x) + x.$$

This formulation makes it easier for the network to learn the identity mapping by setting $F(x) = 0$, and it improves gradient flow during backpropagation.

As a result, ResNets can be trained with significantly greater depth compared to standard convolutional networks without suffering from degradation in performance due to optimization difficulties. By stacking many convolutional layers in this manner, ResNets are able to extract rich and hierarchical feature representations from image input data.

2.1.4 Knowledge Distillation

Knowledge distillation refers to the process of transferring knowledge from a larger, more complex model to a smaller model. In many cases, the larger model has high capacity and achieves strong performance, but it may be inefficient for deployment due to its computational cost [12]. Using a smaller model for the same task can therefore be beneficial, especially during inference where efficiency is important.

Instead of training only on ground-truth labels, the smaller model is trained using both the ground-truth labels and the output of the larger model. By learning to mimic the output distribution of the larger model, the smaller model can capture additional information about the problem, such as relationships between classes [12]. This often leads to better performance compared to training the smaller model directly on ground-truth labels alone.

2.2 Attention

When developing models for reasoning over 3D scenes or other structured data, it is important to capture relationships between different parts of the input. In natural language processing, sentences are typically decomposed into tokens, each of which is mapped to a feature representation. Similarly, in 3D scene understanding, image data can be divided into patches or regions that are encoded into feature vectors.

Attention mechanisms allow feature representations to relate to each other by letting elements incorporate information from other parts of the input. This enables information to be exchanged across the data, helping the model capture relationships and improve the learned representations.

In attention layers, each input feature vector is linearly transformed into three distinct representations using learned mappings W_q, W_k, and W_v. These correspond to queries, keys, and values, respectively.

The attention mechanism computes interactions between elements by comparing queries with keys, and uses the resulting similarities to weight and aggregate the corresponding values. In this way, each element can incorporate information from other elements and refine its representation.

In practice, attention is computed over a set of queries simultaneously, which are stacked into a matrix Q. Similarly, keys and values are stacked into matrices K and V. The attention operation can then be written as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where d_k denotes the dimensionality of the key vectors [23].

In the following, we introduce attention mechanisms relevant for this work.

2.2.1 Self-Attention & Cross-Attention

When attention is applied between feature vectors within the same set, it is referred to as self-attention. In this case, each element updates its representation by attending to all other elements in the same set. For example, feature vectors corresponding to image patches can attend to each other, allowing the model to capture spatial relationships and distribute information across the image.

In contrast, attention can also be applied between two different sets of feature vectors. In the original Transformer architecture, this is referred to as encoder-decoder attention [23], where queries are taken from the decoder and keys and values from the encoder. In this work, we will refer to this mechanism as cross-attention. This allows elements from one representation to incorporate information from another.

For example, feature vectors representing regions in 3D space can attend to feature vectors extracted from images of the same scene.

In both cases, the feature vectors are linearly transformed using learned mappings to produce queries, keys, and values. The key distinction is whether these transformations are applied to a single set of features (self-attention) or to two different sets (cross-attention).

2.2.2 Multi-Head Attention

Instead of performing attention using a single set of queries, keys, and values, multi-head attention projects the input features into multiple distinct subspaces using separate learned linear mappings [23]. Attention is then applied independently in each of these subspaces, referred to as attention heads. Each head produces a representation by attending to the input features, and the resulting outputs are concatenated and linearly transformed to form the final representation. This allows the model to capture different types of relationships in parallel, as each head can focus on different aspects of the data.

2.2.3 Deformable Attention

In the general formulation of attention, each feature vector can attend to all others through the interaction of queries, keys, and values. While this enables the modelling of global dependencies, it also leads to a computational cost that grows with the number of tokens. Moreover, it is not always necessary for a feature vector to attend to all others. For example, a feature representing a point in 3D space is unlikely to benefit equally from all image regions observing the scene. To address this, it is advantageous to restrict attention to a smaller set of relevant locations. This is referred to as "Deformable Attention".

In deformable attention, each query attends only to a small set of spatial locations rather than all possible keys [28]. These locations are defined relative to a reference point associated with the query. Instead of using a fixed sampling pattern, the model learns both the offsets from the reference point and the corresponding attention weights. This allows each query to adaptively select and aggregate information from the most relevant regions of the input.

In the context of this work, the queries correspond to feature vectors in 3D space. Each query is associated with a reference point in the image plane, and attends to a small set of locations in the image feature maps extracted from the input views. The sampled features are then aggregated and used to update the query representations, enabling them to better capture the structure and appearance of the scene.

2.2.4 Positional Encoding

In attention mechanisms, feature vectors can interact and exchange information with each other. However, these feature vectors do not contain any information about their spatial location by default. This means that, without additional information, the model cannot distinguish between where features are located in an image or in 3D space.

To address this, positional encodings are introduced. These are additional vectors that encode the spatial position of each feature and are added to the feature representations [23]. By including this information, the attention mechanism can take spatial relationships into account when comparing and combining features.

2.3 Camera Geometry

2.3.1 Projective Geometry

In order to model a 3D scene from image data, it is necessary to understand how points in three-dimensional space are projected onto a two-dimensional image plane. This process is described by projective geometry.

A point in $\mathbb{R}^3$ with coordinates (X, Y, Z) can be projected onto the image plane as

$$(x, y) = \left(\frac{X}{Z}, \frac{Y}{Z}\right),$$

assuming a simple pinhole camera model with the camera center at the origin. This formulation shows that the projection depends on the depth Z, meaning that points further away appear closer together in the image.

2.3.2 Homogeneous Coordinates

In computer vision, homogeneous coordinates are commonly used to represent points and transformations. They allow projective transformations to be expressed in matrix form.

A point in $\mathbb{R}^3$ with coordinates (X, Y, Z) can be represented in homogeneous coordinates as $(X, Y, Z, 1)$. Similarly, a 2D point (x, y) can be represented as $(x, y, 1)$. More generally, a point in homogeneous coordinates is defined up to a nonzero scale factor, meaning that

$$(X, Y, Z, 1) \sim (\lambda X, \lambda Y, \lambda Z, \lambda), \quad \lambda \neq 0.$$

This representation allows perspective projection to be written as a linear operation followed by a normalization step. After applying a transformation, the Cartesian

coordinates are recovered by dividing by the last component. Homogeneous coordinates therefore provide a convenient framework for describing camera projections and coordinate transformations using matrix operations.

2.3.3 Camera Model

Using the concepts from the previous sections, the projection between 3D world coordinates and 2D image coordinates can be described using a pinhole camera model. This model is defined by intrinsic and extrinsic parameters.

The intrinsic matrix K describes the internal properties of the camera, such as focal length and principal point, and maps coordinates from the camera frame to pixel coordinates. It can be written as

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

where f_x and f_y denote the focal lengths and (c_x, c_y) the principal point.

The extrinsic parameters describe the transformation of a 3D point X_{world} from the global coordinate system to the camera coordinate system. They consist of a rotation matrix R and a translation vector t, which together define the transformation

$$X_{cam} = RX_{world} + t.$$

Combining these components, the full projection from a 3D point in homogeneous coordinates to pixel coordinates can be expressed as

$$x \sim K[R \mid t]X,$$

where X is a point in the world coordinate system and x is the corresponding point in the image. Both X and x are expressed with homogenous coordinates.

By reversing this transformation by providing depth information, it is also possible to map image points back into 3D space. This is particularly important in multi-view settings, where information from multiple camera images is used to reconstruct and reason about a 3D scene.

2.4 Gaussian Scene Representation

In this work, 3D Gaussians are used to form an explicit scene representation. Each Gaussian models a local region in space and is defined by a mean position $\mu \in \mathbb{R}^3$ and

a covariance matrix $\Sigma \in \mathbb{R}^{3\times3}$. The spatial influence of a Gaussian can be described as

$$G(x) = \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right),$$

which assigns higher values to points closer to the mean.

The covariance matrix Σ determines the shape and orientation of the Gaussian. It can be decomposed into a rotation matrix R and a scaling matrix S as

$$\Sigma = RSS^TR^T.$$

This formulation allows each Gaussian to take the form of an anisotropic ellipsoid, meaning it can stretch differently along different directions.

Representing a scene as a collection of such Gaussians results in a sparse 3D representation, where each Gaussian captures a local part of the scene. Due to their anisotropic nature, Gaussians can model structures of varying shapes and sizes, and combining many Gaussians enables a flexible and efficient approximation of complex geometry [17].

To enrich the Gaussians with additional information, each Gaussian is also assigned an opacity value α and a feature vector f_G of arbitrary dimension. The feature vector encodes appearance information, such as color or learned features, which are used during rendering.

A 3D scene represented in this way can be rendered into 2D camera space by projecting the Gaussians onto the image plane. This enables self-supervised learning, where a 3D representation is optimized by rendering images from known camera views and comparing them to the corresponding input images.

This rendering process is referred to as Gaussian splatting. Each 3D Gaussian is projected into the image, where it forms a 2D Gaussian distribution. The Gaussian means μ are projected into the image together with their covariances. The projected covariance becomes

$$\Sigma' = JW\Sigma W^TJ^T,$$

where J is the Jacobian of the perspective projection, given by

$$J = \begin{bmatrix} f_x/z & 0 & -f_xx/z^2 \\ 0 & f_y/z & -f_yy/z^2 \\ 0 & 0 & 0 \end{bmatrix},$$

and where W is the world-to-camera transformation matrix, with f_x and f_y denoting the focal lengths of the camera [26].

The contribution of each Gaussian to a pixel's feature vector is determined by both its learned opacity and its spatial influence in the image plane. Specifically, each

Gaussian has an associated opacity parameter α_i, which is modulated by its projected 2D Gaussian distribution. This means that the effective opacity contribution at a pixel depends on the distance from the projected Gaussian mean, with values decreasing according to the Gaussian falloff.

The final per-pixel feature vector is obtained by accumulating contributions from multiple Gaussians using alpha blending along the viewing ray. This is expressed as

$$C = \sum_{i=1}^{N} T_i \alpha_i c_i,$$

where c_i is the feature vector of the i-th Gaussian, α_i is its opacity contribution at that pixel, and

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

represents the accumulated transmittance of previous Gaussians along the viewing ray, ordered from front to back based on depth [17].

In addition to per-pixel feature vectors, Gaussian splatting can also produce a per-pixel depth map by accumulating depth values along the viewing ray. This assigns each pixel a depth based on the contributions of the Gaussians along that ray [26].

This splatting process enables efficient and differentiable rendering of a 3D scene, making it well suited for optimization from image data.

2.5 Vision Foundation Models

In recent work, it is common to leverage Vision Foundation Models (VFMs). VFMs are large-scale models trained on extensive datasets, typically in a self-supervised or weakly supervised manner, to learn general-purpose visual representations. These representations can be applied to a wide range of downstream computer vision tasks, often with little or no additional training, and can be further fine-tuned when task-specific adaptation is required.

One key advantage of VFMs is their ability to produce rich and transferable feature representations that capture high-level semantic information from images. This makes them particularly useful in settings where labelled data is limited or where a strong prior representation is beneficial.

Two VFMs commonly used in prior work, and also relevant to this work, are CLIP [20] and Metric3Dv2 [13], which are used to extract feature maps and depth information, respectively. CLIP is trained on large-scale image–text pairs, learning a joint embedding space in which images and their corresponding textual descriptions are mapped to similar feature representations. As a result, visual features corresponding

to semantic concepts, such as objects, can be aligned with their textual counterparts. For example, feature vectors corresponding to cars can be identified by comparing them to the embedding of the word "car". This makes CLIP particularly suitable for evaluation on object classes.

Metric3Dv2 is a model designed to estimate metric depth from a single input image. It predicts a depth value for each pixel, producing dense depth maps with high accuracy. These depth estimates can be used as a strong geometric prior, providing reliable approximations of scene structure.

Despite their versatility, VFMs also exhibit limitations. Since they are trained on specific datasets, their learned representations may reflect biases present in the training data. As a result, their performance can degrade when encountering objects and scenarios that were not well represented during training. For example, a model trained on older datasets may struggle to produce meaningful features for newly introduced object categories.

2.6 Losses and Metrics

2.6.1 IoU and mIoU

Intersection-over-Union (IoU) is a common evaluation metric for segmentation and occupancy prediction tasks. It measures the overlap between predicted and ground-truth regions by comparing their intersection to their union:

$$\text{IoU} = \frac{TP}{TP + FP + FN},$$

where TP, FP, and FN denote the numbers of true positives, false positives, and false negatives, respectively [15].

For semantic segmentation tasks involving multiple classes, the mean Intersection-over-Union (mIoU) is commonly used. mIoU is computed by evaluating the IoU independently for each semantic class and averaging the result across classes:

$$\text{mIoU} = \frac{1}{|C'|} \sum_{i \in C'} \frac{TP_i}{TP_i + FP_i + FN_i},$$

where C' represents the set of semantic classes, and TP_i, FP_i, and FN_i denote the true positives, false positives, and false negatives for class i, respectively [15].

IoU and mIoU evaluate the overlap between predicted and ground-truth regions, making them well suited for semantic occupancy prediction.

2.6.2 L1 and Structural Similarity

Two relevant loss functions for this work are the L1 loss and the structural similarity index (SSIM). Both can be used to measure the difference between two images in RGB space.

The L1 loss is defined as

$$\mathcal{L}_{\text{L1}} = \frac{1}{N}\sum_{i=1}^{N}|x_i - y_i|,$$

where x_i and y_i denote corresponding pixel values in the two images, and N is the total number of pixels. This loss measures the absolute difference between pixel intensities.

In contrast, SSIM [25] focuses on perceptual similarity by comparing local image structure. It is computed over small windows x and y and is defined as

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)},$$

where μ_x and μ_y denote the local means, σ_x^2 and σ_y^2 the variances, and σ_{xy} the covariance between the two image patches. The constants C_1 and C_2 are used for numerical stability. We use 7x7 sliding image windows, resulting in an SSIM map with the same dimension as the images.

For each pair of image patches we have:

$$-1 \leq \text{SSIM}(x, y) \leq 1$$

Structual similarity loss is constructed for each location as:

$$\mathcal{L}_{\text{SSIM}}(x, y) = 1 - \text{SSIM}(x, y),$$

The final loss $\mathcal{L}_{\text{SSIM}}$ is the average of $\mathcal{L}_{\text{SSIM}}(x, y)$ for all (x, y) pairs.

While the L1 loss measures pixel-wise differences, SSIM captures structural information such as contrast and luminance, making it more aligned with human visual perception.

2.6.3 Scale-Invariant Log Loss

Scale-invariant logarithmic loss (SILog) is commonly used for depth estimation as an alternative to standard L1 loss [8]. Given predicted depths y_i and ground-truth depths y_i^*, the SILog loss is defined as

$$\mathcal{L}_{\text{SILog}} = \frac{1}{N}\sum_{i=1}^{N} d_i^2 - \frac{\lambda}{N^2}\left(\sum_{i=1}^{N} d_i\right)^2,$$

where

$$d_i = \log y_i - \log y_i^*,$$

and $\lambda \in [0, 1]$ controls the degree of scale invariance.

By operating in logarithmic space, the loss becomes more sensitive to relative depth differences than absolute metric differences. This reduces the influence of large depth values and places greater emphasis on preserving the geometric structure of the scene. In our case it can be used to calculate the difference between the predicted depth in a pixel, and the true depth of that pixel.

2.6.4 Cosine Similarity Loss and Principal Component Analysis

To measure the similarity between two feature vectors, cosine similarity is commonly used. Given two feature vectors x and y, the cosine similarity is defined as

$$\cos(x, y) = \frac{x \cdot y}{\|x\|\|y\|},$$

where $x \cdot y$ denotes the dot product and $\|\cdot\|$ the Euclidean norm. This corresponds to the cosine of the angle between the two vectors in feature space, and measures their directional alignment independently of their magnitude.

Cosine similarity can be formulated as a loss function by minimizing the negative similarity, for example

$$\mathcal{L}_{\cos} = 1 - \frac{x \cdot y}{\|x\|\|y\|},$$

which is minimized when the vectors are aligned.

When working with high-dimensional feature representations, such as those obtained from vision foundation models, it is often beneficial to reduce the dimensionality of the feature space. Many dimensions may contain redundant or less informative variations.

Principal Component Analysis (PCA) is a common technique for this purpose. PCA identifies a set of orthogonal directions, referred to as principal components, that capture the largest variance in the data. By projecting feature vectors onto a reduced set of these components, it is possible to retain the most informative structure of the data while discarding less relevant variations. Cosine similarity can then be computed in this reduced feature space, providing a more efficient and compact representation while preserving the most important relationships between features.

2.6.5 Weighted Cross-Entropy Loss

Cross-entropy loss is a commonly used objective function for classification tasks. It measures how well the predicted class probabilities align with the ground-truth labels. For a single sample, the cross-entropy loss is defined as

$$\mathcal{L}_{CE} = -\sum_{c} y_c \log(p_c),$$

where y_c denotes the ground-truth label for class c, typically represented using a one-hot encoded vector, and p_c is the predicted probability for class c.

In many practical settings, the class distribution is imbalanced, meaning that some classes occur significantly more frequently than others. This is particularly common in semantic occupancy prediction. To mitigate this imbalance, a weighted version of the cross-entropy loss can be employed:

$$\mathcal{L}_{WCE} = -\sum_{c} w_c \, y_c \log(p_c),$$

where w_c is a class-specific weighting factor. By assigning larger weights to underrepresented classes, the model is encouraged to place greater emphasis on these classes during training. This reduces the bias toward dominant classes and can lead to more balanced semantic predictions.

2.6.6 Lovász-Softmax loss

The Lovász-Softmax loss [2] is a differentiable surrogate for the Jaccard loss, defined as one minus the Intersection-over-Union (IoU) score. It can be used for semantic segmentation tasks where evaluation is commonly based on mIoU.

Direct optimization of the IoU metric is difficult since it depends on discrete region overlaps and class assignments. The Lovász-Softmax loss addresses this by constructing a continuous approximation of the IoU objective from the softmax probabilities produced by the network.

Instead of only considering individual voxel or pixel predictions, the loss accounts for how prediction errors affect the overlap between predicted and ground-truth semantic regions. In the multiclass setting, the loss is computed independently for each semantic class and averaged similarly to the mIoU evaluation metric.

This makes the Lovász-Softmax loss well suited for semantic occupancy prediction, where performance is commonly evaluated using mIoU.

2.6.7 Geometric Consistency Warping

In addition to comparing images and feature representations directly, it is also possible to enforce consistency through the underlying scene geometry. This idea is used in Monodepth2 [10], where geometric constraints are used to learn depth without explicit supervision.

Given an image and a corresponding depth map, each pixel can be projected into 3D space using camera intrinsics. By applying the relative camera transformation between different timesteps, this 3D point can be reprojected into another view, establishing correspondences between pixels in the two images. This allows one image to be warped into the viewpoint of another. Figure 1 illustrates the same physical spot, as seen from different frames.

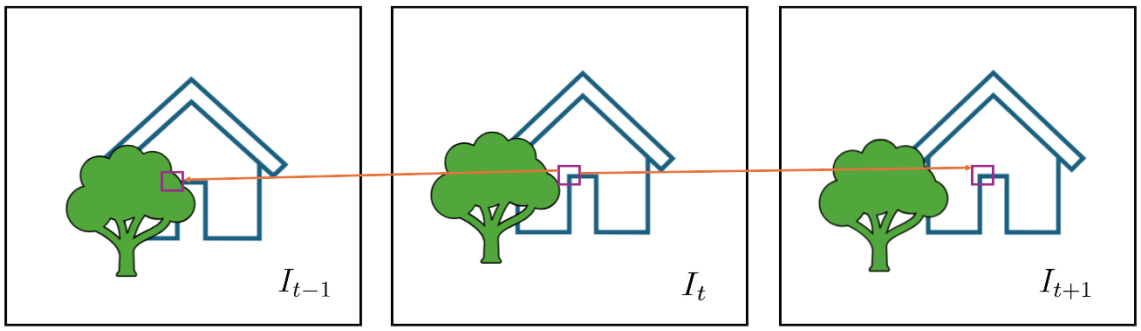


Figure 1: Example a region of an image, seen from different time frames. Note that regions visible in the current frame are not necessarily visible in adjacent frames due to occlusions and viewpoint changes.

If the estimated depth is accurate, the warped image should be consistent with the target image. This consistency can be measured using losses such as L1 and SSIM, as described previously. In this way, the warping operation enables comparison of corresponding pixels across views rather than only within a single image.

This provides a geometric constraint on the model, as incorrect depth predictions lead to misalignment between warped and target images. Minimizing this discrepancy encourages depth estimates that are consistent with the observed scene geometry.

Occlusions, such as the one visualized between $t-1$ and t in figure 1, can cause high loss even in cases where the depth is correct.

3 Dataset

When training a vision model for AD it is important to use good data. We use the widely adopted nuScenes [4] dataset because of its vast and structured collection of driving images. Additionally, nuScenes is commonly used in AD-research, making it suitable for benchmarking and comparison with other published models.

3.1 Structure of nuScenes

The nuScenes [4] dataset consists of sensor data collected across 1000 driving scenes from Boston and Singapore. Sensor data from cameras, LiDAR and RADAR are included from all scenes. Our model only uses the camera images as input. Camera instrinsics and extrinsics are known and included in nuScenes. In nuScenes, the data collection vehicle is equipped with six outward facing cameras, which all capture images at 12 Hz. An overview of the car and its sensors is shown in figure 2. At 2 Hz, the sensor data is collected into keyframes, of which there are 40000. Keyframes are annotated with 23 object classes. Importantly for this work, keyframes also include the ego pose of the vehicle. Since we need ego pose for training, and annotations for evaluation, we only use keyframe images. NuScenes is split up into 700 scenes for training, 150 for validation and 150 for testing, with each scene lasting 20 seconds. The test dataset is not publicly available since it is reserved for benchmarking. Therefore we, following common practice, report results on the validation dataset.

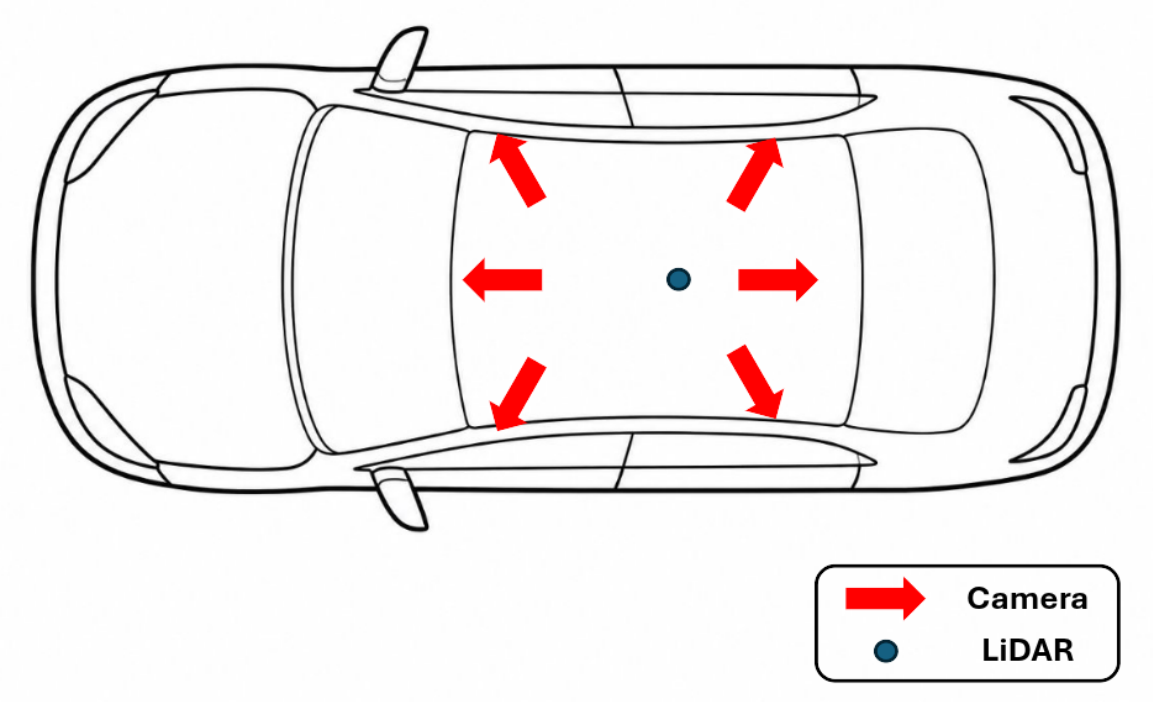


Figure 2: Sketch of the nuScenes car.

3.2 Occ3D-nuScenes

While our method aims to be as self-contained as possible, we still need an evaluation benchmark. For this, we use Occ3D-nuScenes [22]. While nuScenes does include multimodel sensor data and annotations, it does not include dense voxel-level occupancy ground truths or semantic labels. Occ3D-nuScenes derives semantic occupancy by aggregating multi-sensor information along with annotations. The result is voxelized semantic occupancy labels for spanning the range [-40m, -40m, -1m, 40m, 40m, 5.4m] with voxel size [0.4m,0.4m,0.4m]. Occ3D-nuScenes contains 16 semantic classes as well as an "others" class. Additionally, Occ3D-nuScenes handles occlusions, so we only perform evaluation on voxels which are visible from the perspective of the vehicle.

The semantic classes included are shown in table 1.

Class	**Count**	**Percentage (%)**
others	944,004	0.17
barrier	1,897,170	0.35
bicycle	152,386	0.03
bus	2,391,677	0.44
car	16,957,802	3.09
construction vehicle	724,139	0.13
motorcycle	189,027	0.03
pedestrian	2,074,468	0.38
traffic cone	413,451	0.08
trailer	2,384,460	0.43
truck	5,916,653	1.08
driveable surface	175,883,646	32.01
other flat	4,275,424	0.78
sidewalk	51,393,615	9.35
terrain	61,411,620	11.18
manmade	105,975,596	19.29
vegetation	116,424,404	21.19

Table 1: Semantic class distribution in Occ3D-nuScenes (training set).

An example of a voxelized scene from Occ3D-nuScenes is shown in figure 3.

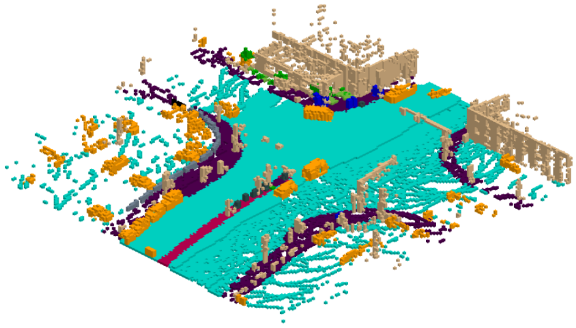


Figure 3: Example of Occupancy ground truth from a scene in Occ3D-nuScenes. Voxels are coloured by semantic class.

4 Method

While developing our method, we made use of the fact that other published works have already implemented some of the components needed in our model. We developed our model by using the published code from GaussTR [16] as a baseline, and building our own modules on top of it. The authors of [16] refer to GaussTR as ”self-supervised”. However, the model relies heavily on VFMs both as backbone and for supervision. In this work we regard this use of VFMs as a form of external supervision which we do not want. GaussTR does show that a Gaussian representation, which is not trained on occupancy ground truth, can work well. This is why we choose it as our baseline. Much of our work is aimed at making the GaussTR-method more self-contained. The following section will outline the model architecture as well as make clear which parts are from GaussTR and which parts are our own contribution.

4.1 GaussTR Baseline

There are a few different configurations in the original GaussTR implementation. The one we use as baseline takes MaskCLIP [7] (implemented with ViT-B/16) features of multi-view images, upsampled with FeatUp [9], which we will refer to as FeatUp features. These are then sent to a neck which converts the FeatUp features from dimension 512 to 256, which is the embedding dimension of the transformer decoder. These features are used as memory for the transformer decoder module, which takes learned query embeddings as input along with a set of 2D reference points in each camera plane. A decoder layer applies deformable cross attention, self attention and lastly a feed-forward layer. In GaussTR, there are three decoder layers in total. The two-dimensional reference points are used for positional encoding of the queries and are refined at every decoder layer. In GaussTR, both queries and reference points

are per-view. There is no attention interaction between queries in different camera views.

After each layer of the transformer decoder, there is a module referred to as the Gaussian head. The Gaussian head consists of a set of MLPs which predict opacity, scale and features for all Gaussians. Rotations are not predicted by the head, but calculated per-view following each camera's rotation relative to ego coordinates. The 3D mean of each Gaussian is determined by the reference points along with the Metric3Dv2 VFM (implemented with ViT-Large) and an MLP which predicts deltas. For simplicity, this VFM will be referred to as Metric3D. GaussTR applies a 2D delta, predicted by the MLP, in each camera view to all reference points. The shifted reference points are then used to sample depth at their corresponding pixel, where depth is predicted by Metric3D. The sampled depth is used to "lift" the Gaussians into 3D. Lastly a depth delta, predicted by the same MLP, is applied to determine the final three dimensional positions of the Gaussians.

After these steps, the training and evaluation paths differ. During training, the Gaussians are splatted into the camera views. At this step, Gaussians do no longer "belong" to a specific camera, so each Gaussian is splatted into every frame where it is visible. The splatting results in per pixel rendered depth and feature maps. GaussTR applies PCA and cosine loss on the rendered features with the input FeatUp features to enforce semantic alignment with MaskCLIP. It also uses the Metric3D depth predictions to supervise the rendered depth maps, to enforce correct geometry. Depth is supervised with the following combination of L_1 and scale-invariant log loss:

$$\mathcal{L}_{\text{depth}} = \mathcal{L}_{\text{silog}} + 0.2\mathcal{L}_{\text{L}_1}$$

In the evaluation path, Gaussians are not splatted. Instead, the model first computes the dot product between each Gaussian's feature vector and the CLIP text embeddings, treating the result as logits and converting them into semantic probabilities. 3D space is then voxelized, and for each voxel, all Gaussians whose centres lie within three standard deviations are considered. These Gaussians contribute their opacities and semantic probabilities, weighted based on the Mahalanobis distance between the voxel centre and the Gaussian mean, so that closer Gaussians have a stronger influence. The contributions are aggregated per voxel, producing a density value and combined semantic probabilities, which are normalized again with a softmax to ensure they sum to one. At this stage, the probabilities correspond to the CLIP-based categories, so they are mapped to the Occ3D-nuScenes labels to enable evaluation. Finally, each voxel is classified as occupied if its density exceeds a chosen threshold, which is treated as a hyperparameter. If a voxel is classed as occupied, it is assigned the semantic label with the highest probability. It is treated as empty otherwise. The GaussTR baseline is summarised in figure 4.

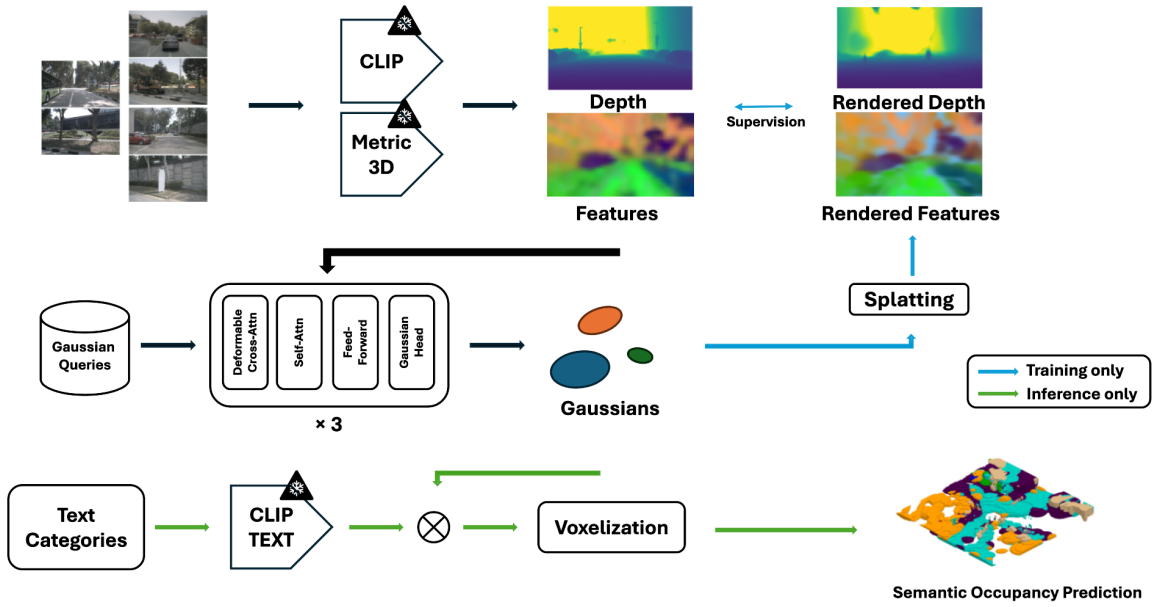


Figure 4: Overview of the original GaussTR pipeline. Frozen VFM modules are marked with a snowflake symbol.

As seen in table 1, Occ3D-nuScenes includes the classes ”others” and ”other flat”. GaussTR never predicts these classes. This is likely because it is difficult to get sufficient alignment between image features and text features for these classes, which are not as well defined visually as the other classes.

4.2 Modifications to GaussTR

As described above, GaussTR relies on two sets of VFM features: FeatUp and Metric3D, both at inference and for supervision. We handle these cases separately, and perform ablations our modifications. This results in different supervision versions of our model, with different uses of VFMs and supervision types, including a fully self-contained and self-supervised version.

4.2.1 ResNet Replaces FeatUp

We replace the pre-computed set of FeatUp features as input to the model with a standard ResNet50 backbone, pre-trained on ImageNet [6]. Since the ResNet features have different dimension and structure to FeatUp, we use a different neck to the one used with FeatUp, to ensure that the features are on the form which the transformer decoder expects.

We also replace FeatUp with ResNet for supervision, to remove dependence on FeatUp completely. Instead of computing cosine loss against FeatUp features, we map the rendered features to the shape of the four ResNet output stages with four convolutional layers. We then compute cosine loss against each ResNet stage and sum the result. Since we use ResNet features as input and as supervision target, we freeze the backbone, meaning that we use standard off the shelf ResNet-50 weights, trained on ImageNet [6]. As an alternative, we allow the backbone to be trainable, but supervising the rendered features with FeatUp instead, thereby performing a form of knowledge distillation from FeatUp to ResNet. Then, we freeze this modified ResNet instead and train with its features as both input and supervision. We compare the performance of the standard frozen ResNet backbone and the distilled ResNet variant, which has inherited knowledge from FeatUp.

4.2.2 Replacing Metric3D

As described, GaussTR uses Metric3D depth to place Gaussians at the correct 3D position from reference points in the camera plane. We simply replace Metric3D depth at this stage with a depth head in the form of an MLP, which takes decoded attention queries as input and outputs predicted depth for every reference point.

Supervising the rendered features is not enough to enforce correct geometry, since there are many different possible 3D structures which result in correct projections in the cameras. To combat this, GaussTR also uses Metric3D depth for depth supervision in GaussTR. We replace this depth loss term with a geometric consistency warping, inspired by [10]. For each pixel in each input image at time t (target), we sample its rendered depth and project it into the same camera view at times $t-1$ and $t+1$ (sources). The RGB source RGB values are bilinearly sampled and warped into target time t. If the rendered depth used for warping and sampling is correct. the target and warped source images should match. We therefore use a warping loss for each pixel p:

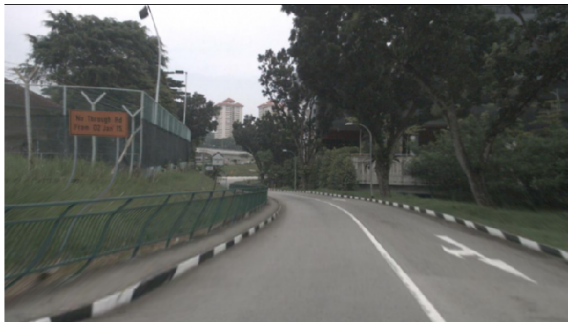
$$\mathcal{L}_{warp}(p) = \lambda \mathcal{L}_{SSIM}(p) + (1-\lambda)\mathcal{L}_{L_1}(p)$$

with $\lambda = 0.85$ like in [10].

The warping is visualized in figure 5. Clear distortions are visible in the warped image due to the ego motion of the vehicle and imperfect rendered depth.

(a) Target image, time t

(b) Source image, time $t-1$

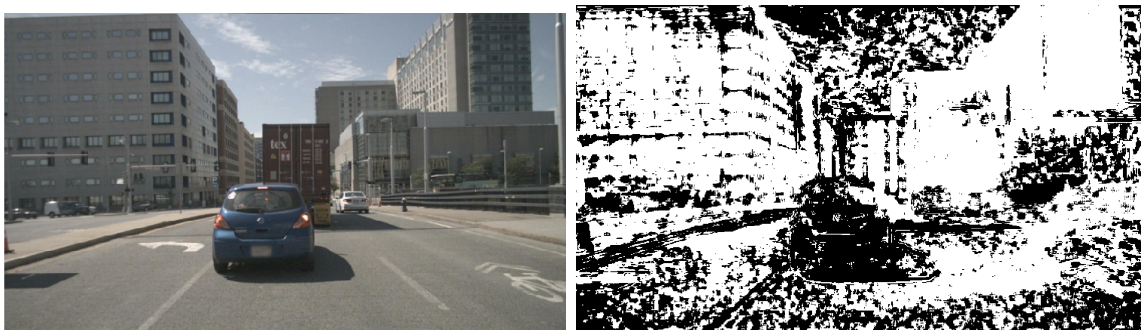


(c) Warped source image

Figure 5: Visualization of warping, where loss is computed between target and warped source image. The source image at time $t+1$ is excluded for simplicity.

Additionally, we also use per-pixel minimum loss as well as auto-masking, which are also used in [10]. Per-pixel minimum means that we only use one of the source images $t-1$ and $t+1$ in each pixel, namely the one which produces the smallest loss in that particular pixel. This is meant to reduce the problem of certain objects being occluded in one time frame but not another. An example of such an occlusion is shown in figure 1, where a tree occludes a certain area at $t-1$ but not at $t+1$. In this example, computing loss for that area for the source image at $t-1$ would not make sense.

Auto-masking means that we mask away those pixels which do not change meaningfully between target and source. This is meant to deal with the fact that this loss assumes a static scene, but in reality objects can move. Ideally, objects moving at similar velocity to the ego vehicle will be masked. Pixels whose minimum loss is smaller between unwarped source and target than between warped source and target are masked, and the loss is not counted. An example of our auto-mask is shown in figure 6.



(a) Target image **(b)** Auto-mask, black pixels not included in loss

Figure 6: Example of target image and auto-mask used in warping depth loss. It is clear that the auto-mask masks a significant portion of the car in front and its shadow, which are moving at similar speed to the ego vehicle. Some other areas of the image are also masked.

Following [10], we also evaluate an edge-aware smoothness loss of the form

$$\mathcal{L}_s = |\partial_x d^*| \, e^{-|\partial_x I|} + |\partial_y d^*| \, e^{-|\partial_y I|}$$

where I denotes the RGB image and $d^* = d/\bar{d}$ is the rendered inverse depth map normalized by its mean value. This loss is based on the assumption that depth discontinuities are more likely to occur at image edges than in locally smooth image regions. We evaluate whether this loss improves model performance and report the corresponding ablation results.

4.2.3 LiDAR Depth Supervision

To aid in the learning of correct depth, we test the effect of using LiDAR point clouds for depth supervision. Since LiDAR is part of the raw nuScenes dataset and does not rely on human-generated labels or annotations, we consider this a form of self-supervision. We project all LiDAR points from a sweep into all cameras where they are visible and calculate their depth per-camera. Per camera, the pixel which a certain LiDAR is projected to is assigned this target depth, which is used to supervise the rendered depth map from Gaussian splatting to ensure correct scene geometry. We try both L_1 and scale invariant logarithmic losses. LiDAR supervision is meant as a complement in the case where geometric consistency warping does not give strong enough depth constraints.

4.2.4 Self-Supervised Pre-Training

As described above, we replace VFM semantic features with ResNet features, and VFM depth with warping, and optionally LiDAR supervision. We test the perfor-

mance of the model with different combinations of self-supervised and VFM components. In the case where we replace all VFM dependencies with the above presented alternatives, we achieve fully self-supervised pre-training. The pipeline of this version of the model is presented in figure 7.

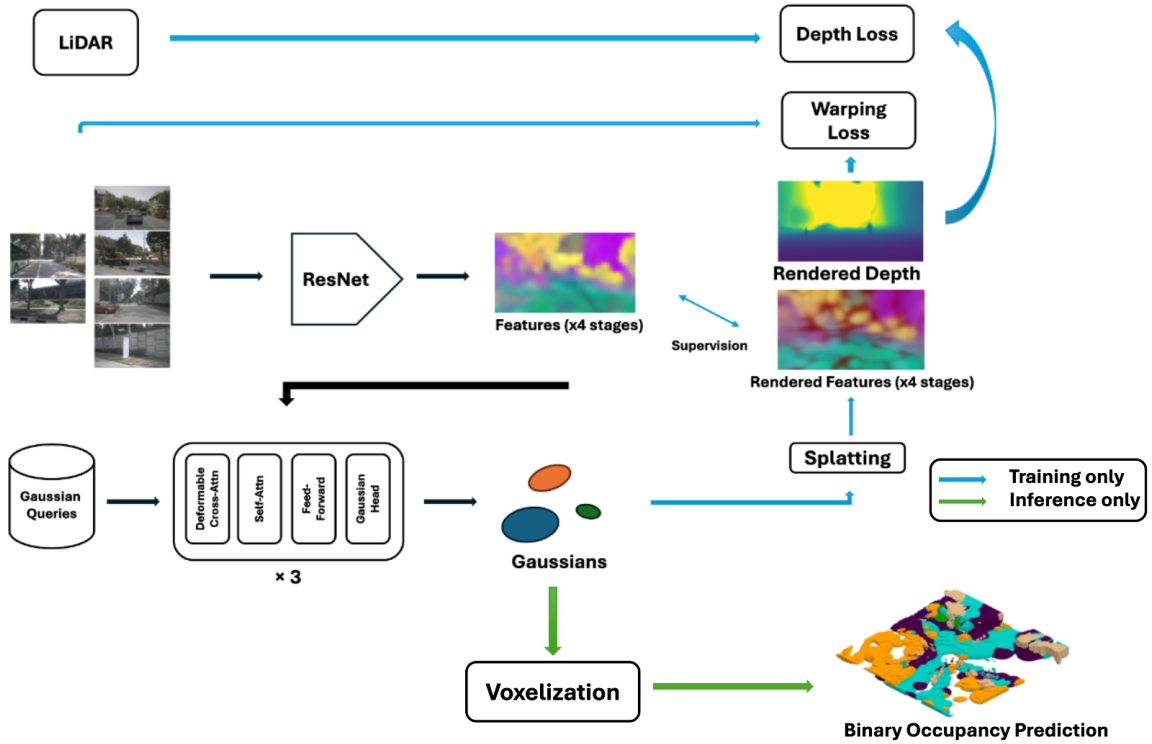


Figure 7: Overview of fully self-supervised pipeline. The ResNet backbone is frozen in some configurations but not in all. Therefore we do not mark it with a snowflake symbol. We clearly state which results were obtained with a frozen backbone and which were not. The validation path results in binary, instead of semantic, occupancy prediction since we no longer have text-CLIP feature alignment. Colours are still included in the voxelized occupancy prediction for clarity.

4.2.5 Maintaining Semantic Separation

Ideally, our method maintains good feature separation of ResNet features, enabling semantic occupancy prediction. However, the zero-shot method used in GaussTR for semantic classification, where voxels are classified according to dot product similarity with CLIP text embeddings is not possible with our ResNet method. This is because we have not trained the Gaussian features to align with CLIP, so the text embedding similarity would not maintain its meaning. Instead we aggregate Gaussian features in voxels. These aggregated voxel features are used as input to a semantic head in the form of an MLP which is supervised on Occ3D-nuScenes, keeping the pre-trained

model frozen. We supervise semantics of all voxels which are occupied in the Occ3D-nuScenes ground truth. Inspired by the supervised occupancy prediction work in [15], we use a combination of weighted cross-entropy and Lovász-softmax loss. As in [15], we weight the Lovász-softmax loss by 0.5:

$$\mathcal{L}_{\text{semantic}} = \mathcal{L}_{\text{CE}} + 0.5\mathcal{L}_{\text{Lovász}}$$

This method is clearly not self-supervised, but it is used to investigate whether good semantic separation is achievable with fully self-supervised pre-training. This semantic tuning is done on half of the Occ3D-nuScenes training set, since the goal of the pre-training is to reduce the need for labelled data. We try the case where the whole pretrained model is frozen, keeping only the semantic head learnable. Additionally, we perform experiments with the feature head of the third layer unfrozen, to enable better feature separation in the Gaussians. None of these two cases impact binary occupancy prediction. This is because binary occupancy is only determined based on accumulated opacity in each voxel, which is not affected by the semantic tuning of either the semantic or feature head.

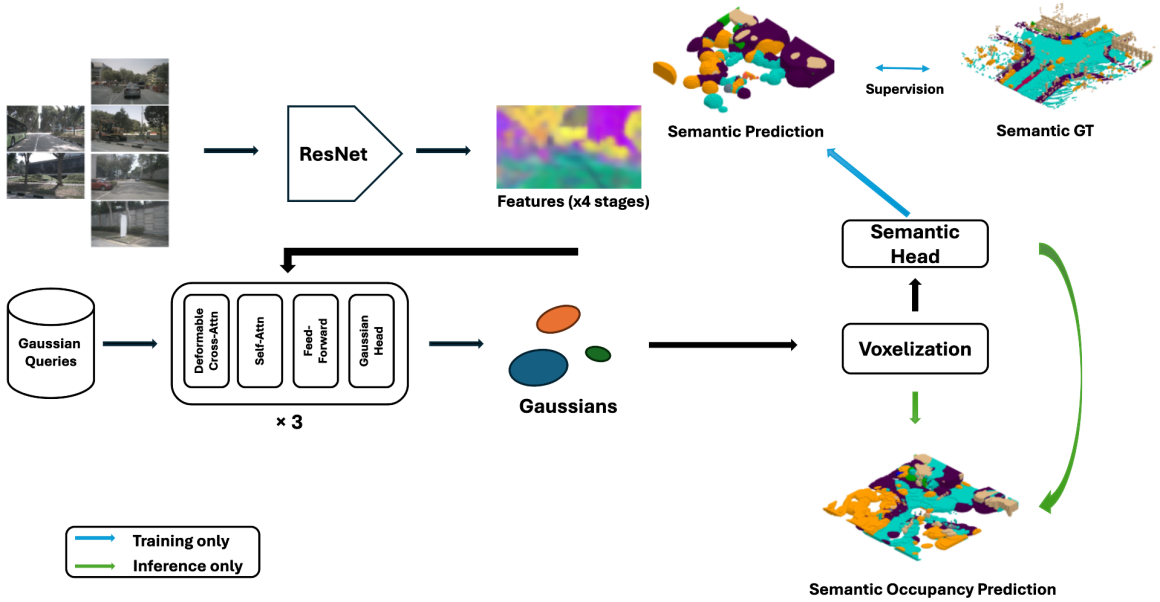


Figure 8: Overview of the semantic tuning pipeline. All components of the model are frozen except the semantic head and, in some cases, the feature head of the third layer. It is made clear which results are obtained with a learnable Gaussian head and which are not.

4.3 Validation

To enable fair comparison with the GaussTR baseline as well as other occupancy prediction models, we also use binary and semantic occupancy prediction as validation tasks. The metrics used to measure performance are Intersection-over-Union (IoU) and mean Intersection-over-Union (mIoU). These metrics are used for binary and semantic occupancy prediction, respectively.

For semantic occupancy prediction, mIoU is computed over the set of non-empty Occ3D-nuScenes classes. We evaluate using the Occ3D-nuScenes camera mask, meaning that only voxels visible to a camera on the vehicle are included in the evaluation.

As described above, occupancy prediction is used as a way of testing general scene understanding. The scene representation should ideally be usable for different downstream tasks, not just occupancy prediction.

4.4 Implementation Details

Like GaussTR we use three transformer decoder layers with embedding dimension 256.

The multiview images are resized to (432, 768) resolution in both the training and evalutation paths before being input to the model. This is slightly smaller than the resolution reported in [16], which is (504, 896), but matches their published code. As in GaussTR [16], we use no random image augmentation.

We train the models presented with the AdamW optimizer, with a weight decay of 5×10^{-3}, on 8 NVIDIA A100 GPUs and batch size 1. We use learning rate 2×10^{-4} and 200 iterations of warmup. After 16 epochs the learning rate is decreased to 2×10^{-5}. We train for 24 epochs but employ early stopping, which means that if validation performance starts to decrease, we report the result from the epoch with the best performance and use its checkpointed weights.

Unless otherwise specified, we have kept hyperparameters as similar as possible to the ones used in the original GaussTR implementation. This includes but is not limited to learning-rate scheduling, number and size of Gaussians, embedding dimensions, and the opacity threshold to determine if a voxel is occupied or not. This was done to enable a fair comparison as possible to the GaussTR baseline.

A notable exception is the Gaussian feature dimension used in the case where ResNet is used to supervise features instead of FeatUp. In this case, the feature of each Gaussian has dimension 64 instead of 512, since there was no longer a need for Gaussians to match the 512-dimensional FeatUp features.

5 Experiments and Results

In this section we present the main results of our experiments. We present a series of different variations of the model, each containing different combinations of the modules we have developed and components from the original GaussTR [16]. The evaluation metrics are occupancy IoU and mIoU on the Occ3D-nuScenes [22] validation set using the camera mask.

First we present the results from GaussTR, both as reported in [16] and the results from our reimplementation, to enable fair comparison. We then proceed to report performance on the different versions of our model, as well as compare with other relevant works. We also demonstrate the results of our semantic tuning on cases where FeatUp was not used as feature supervision. In all cases we use early stopping and present the result of the checkpoint which which scored the highest on the IoU metric on the validation data. IoU and mIoU are reported as percentages.

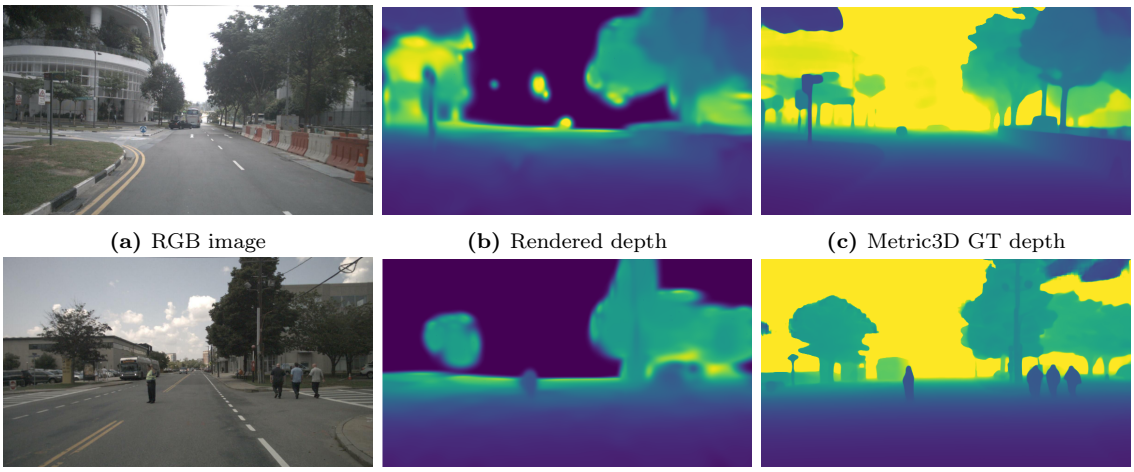
5.1 GaussTR Baseline Performance

Table 2 reports the results presented in [16] as well as our reimplementation. [16] does not present IoU results for the categories "others" and "others flat", which their model is not capable of identifying. However, we choose to include these figures for clarity, since they are included in the calculation of mIoU for fair validation on Occ3D-nuScenes. When evaluating the effectiveness of the modules we develop, we will mainly compare results to those of our reimplementation of GaussTR, to keep the training and validation environments as similar as possible. For the simplest version of GaussTR which only uses FeatUp and Metric3D, only IoU and mIoU are reported. IoU per class is not included. Since this is the version we use as our baseline, those are the numbers we report.

Method	**IoU**	**mIoU**	barrier	bicycle	bus	car	cons. veh.	motorcycle	pedestrian	traffic cone	trailer	truck	drive. surf.	sidewalk	terrain	manmade	vegetation	others	other flat
GaussTR (paper)	44.73	10.76	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
GaussTR (reimp.)	46.32	11.09	2.42	4.08	12.62	18.87	5.82	3.46	2.50	3.09	1.14	13.38	40.22	15.87	22.52	20.09	22.52	0	0

Table 2: Baseline occupancy prediction results on the Occ3D-nuScenes validation set. The results reported for our reimplementation were from the checkpoint which gave maximum IoU.

Our reimplementation of the GaussTR baseline slightly outperforms the published results, as seen in table 2.



(a) RGB image **(b)** Rendered depth **(c)** Metric3D GT depth

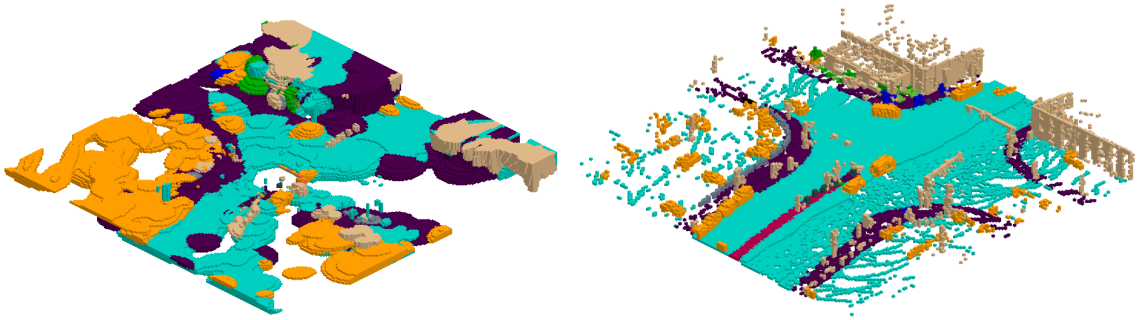
(d) RGB image **(e)** Rendered depth **(f)** Metric3D GT depth

Figure 9: Two sets of rendered and GT depth maps along with corresponding RGB images from our reimplementation of GaussTR. Both examples correspond to the front camera view.

Due to the implementation of depth supervision used in [16], opacity of Gaussians which project to a pixel where the Metric3D ground truth depth is bigger than the set depth limit (51.2 m), is encouraged to be 0. Gaussians with 0 opacity contribute no rendered depth, which is why the rendered depth tends to be 0 for such regions in figure 9. Since evaluation is performed in a volume of size [-40m, -40m, -1m, 40m, 40m, 5.4m], the furthest point away from the vehicle which is evaluated is at $56.8m > 51.2m$. Therefore GaussTR is unable to represent the far corners of the voxel grid. However, these areas are not often included in evaluation, since evaluation is only performed on voxels which are visible to the vehicle.

In our self-supervised versions, we do not have access to Metrid3D ground truths, so we do not get this effect.

Figure 10 shows semantic occupancy prediction as well as Occ3D ground truth for a chosen frame. It looks as though ”driveable surface” is the easiest class to identify, which is also reflected in table 2.



(a) Occupancy prediction **(b)** Occ3D-nuScenes GT occupancy

Figure 10: Visualization of occupancy prediction from our reimplementation of GaussTR and corresponding occupancy GT generated by Occ3D-nuScenes.

5.2 Replacing FeatUp

Method	**IoU**	**mIoU**	barrier	bicycle	bus	car	cons. veh.	motorcycle	pedestrian	traffic cone	trailer	truck	drive. surf.	sidewalk	terrain	manmade	vegetation	others	other flat
RN BB	45.42	9.71	2.00	1.08	9.63	19.53	2.69	0.03	0.42	0.02	1.98	11.43	41.31	16.03	16.06	21.10	21.80	0.00	0.00
RN BB+sup	45.15	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
D RN BB+sup	46.70	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 3: Occupancy prediction results on the Occ3D-nuScenes validation set using ResNet (RN) backbone (BB) with FeatUp supervision, as well as with ResNet supervision (sup). Distilled ResNet marked with D. All cases use Metric3D depth at inference and supervision.

The first row in table 3 shows performance when we replace FeatUp as backbone, but keep it as supervision. Performance decreases slightly from our reimplementation of GaussTR, but IoU is still close to the level of the original version, while mIoU suffers slightly more. It is unsurprising that IoU suffers less from this change than mIoU, since binary occupancy can be learnt through geometry only, while semantic classification needs good feature separation. In this version of the model, the ResNet backbone is learnable, so it can adapt to be able to mimic FeatUp. The checkpoint of this tuned ResNet is the one we mean when referring to the ”distilled’ ResNet backbone.

As described, semantic classification is not possible at the pre-training stage in the case where we also replace FeatUp with ResNet at supervision. Therefore we only report IoU. As described, we use Gaussian feature dimension 64 in these cases, as we no longer need to align with 512 dimensional FeatUp features.

Performance appears to drop slightly when using the non-distilled ResNet, but it is still high at 45.15. When using distilled ResNet, the model even outperforms the original GaussTR, with an IoU of 46.70. Results clearly shows that performance when using ResNet increases when the backbone gets distilled information from FeatUp. The fact that this case outperforms the original FeatUp case indicates that using 512-dimensional Gaussian features may be excessive for performing binary (non-semantic) occupancy prediction.

5.3 Replacing Metric3D

Method	**IoU**	**mIoU**	barrier	bicycle	bus	car	cons. veh.	motorcycle	pedestrian	traffic cone	trailer	truck	drive. surf.	sidewalk	terrain	manmade	vegetation	others	other flat
M3D sup	43.22	9.96	2.11	3.93	11.24	16.38	4.35	3.00	1.49	2.11	0.93	11.41	39.82	14.94	21.69	16.92	18.96	0.00	0.00
Warp	36.22	6.74	1.27	2.02	4.18	8.77	4.37	0.67	0.43	1.10	0.70	5.84	26.74	11.12	19.65	14.40	13.25	0.00	0.00
Warp+smooth	34.63	6.24	1.82	1.88	3.87	7.50	4.13	0.61	0.48	0.31	0.20	6.08	26.53	11.90	18.96	10.74	11.03	0.00	0.00

Table 4: Occupancy prediction results on the Occ3D-nuScenes validation set using FeatUp as input and for supervision. In one case (M3D sup) Gaussians are placed with the depth MLP instead of Metric3D depth, while Metric3D is still used for supervision. The next case (Warp) supervises depth with geometric consistency warping, not using Metric3D at all. The third case (Warp+smooth) uses warping and also includes the edge-aware smoothing loss.

Table 4 shows that IoU performance drops to 43.22 when using a learned depth MLP to place Gaussians in 3D, instead of Metric3D ground truth depth. This is down from 46.32 in our reimplementation of GaussTR, as shown in table 2. mIoU suffers slightly as well. It is unsurprising that losing access to GT depth at inference hurts performance. The model still retains IoU > 43, which indicates that the depth MLP is capable of learning meaningful depth.

In the case where we do not use Metric3D at all, replacing the supervision signal with geometric consistency warping, IoU decreases to 36.22, and mIoU decreases further. This indicates that the self-supervised warping loss does not provide as strong depth supervision as Metric3D ground truth.

The "Warp+smooth" case in table 2 shows the result of adding edge-aware smoothing loss to help depth learning. As seen in the table, both IoU and mIoU decreases. We consistenly saw a slight drop off in performance when including this loss term. Therefore, we do not include it in the experiments presented from now on.

5.4 No VFMs at Inference

One particularly interesting middle ground between the VFM-dependent GaussTR-method and our goal of a fully self-contained model is the case where VFMs are used for supervision but not at inference. We perform two experiments, where one uses both FeatUp and Metric3D supervision. The other experiment supervises features with the original (non-distilled) ResNet, while keeping Metric3D depth supervision. The results are presented in table 5.

Method	**IoU**	**mIoU**	barrier	bicycle	bus	car	cons. veh.	motorcycle	pedestrian	traffic cone	trailer	truck	drive. surf.	sidewalk	terrain	manmade	vegetation	others	other flat
FeatUp + M3D sup	43.99	10.73	2.30	6.28	15.53	19.35	5.71	0.79	0.57	2.79	0.95	11.88	41.07	16.76	20.68	18.45	19.25	0.00	0.00
ResNet + M3D sup	40.30	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 5: Occupancy prediction results on the Occ3D-nuScenes validation using ResNet backbone and depth MLP to place Gaussians. Both cases uses Metric3D depth supervision. One uses FeatUp and the other uses ResNet for depth supervision.

We see good performance in the case with both FeatUp and Metric3D supervision. It even outperforms "M3D sup" from table 4 for both IoU and mIoU. The only difference is that "M3D sup" also uses FeatUp features as input at inference. This indicates that FeatUp as input is not necessary, and the model can be simplified with no loss of performance by using a ResNet backbone instead.

The case in table 5 which uses ResNet for feature supervision suffers slightly, even if it still achieves IoU > 40. The fact that binary IoU drops shows that good feature supervision helps with learning geometry, even though the binary occupancy prediction does not depend on the feature vector itself of each Gaussian. When replacing FeatUp supervision with ResNet, we also lose the ability for zero-shot semantic classification.

These results are particularly interesting since they show that it is possible to not use any VFM at inference for satisfactory performance. This has potential to enable lower latency, thereby making the model more suitable for deployment in a real driving scenario. We do not perform latency tests in this work. However, GaussianFlowOcc [3] also uses a ResNet-50 backbone, with VFMs only for supervision. They report significant improvement in inference speed over GaussTR and cite the lack of VFMs at inference, and the relatively lightweight ResNet-50 backbone, as reasons for this.

5.5 Self-Supervision

In this section we present the results of five different versions of our model. We report results of five different configuration, which do not use VFMs for inference or super-

vision. We include versions with both the original and distilled ResNet backbones, as well as with and without L_1 LiDAR supervision. We do not present results with scale-invariant logarithmic LiDAR depth loss, since results were generally worse than those with L_1 loss.

Method	**IoU**
Base	34.63
Base D	35.79
Base L	34.36
Base D+L	39.52
Only L	33.31

Table 6: Occupancy prediction results on the Occ3D-nuScenes validation set using ResNet backbone and supervision, as well as depth MLP to place Gaussians and warping to supervise depth. Versions with distilled ResNet marked with D and versions with LiDAR depth supervision marked with L. "Only L" has no warping loss, so only LiDAR is supervising depth.

Table 6 shows slightly better performance in the cases with distilled ResNet than original ResNet. However, the most notable result is that LiDAR supervision does not help IoU performance at all with original ResNet, but helps significantly in the case which has distilled ResNet. This is an indication that the higher feature quality from the distilled ResNet is important for the model to be able to learn correct spatial understanding from LiDAR supervision.

"Only L" performs the worst, indicating that warping loss is helpful even when including LiDAR depth supervision.

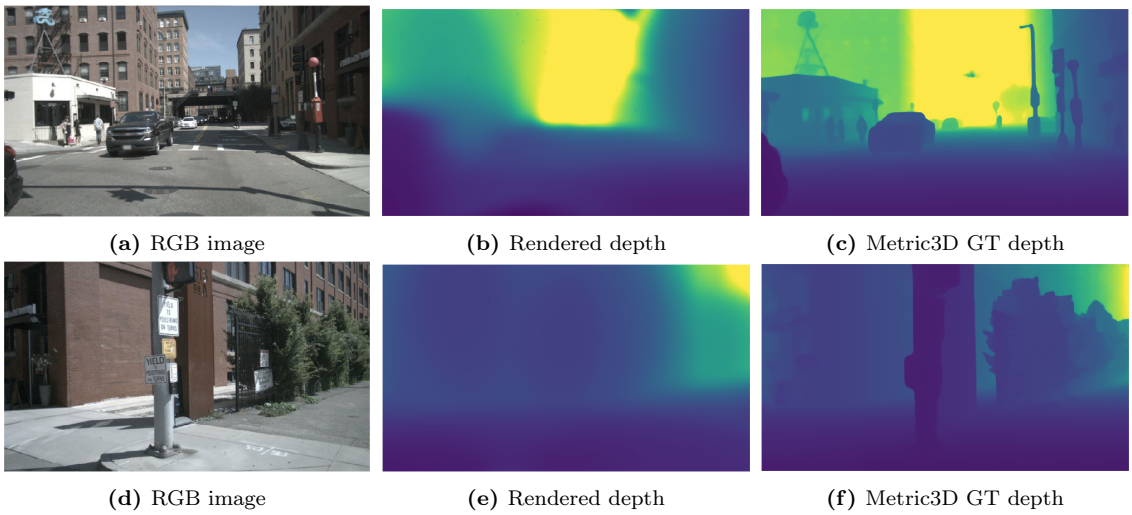
5.6 Number of Gaussians and Scale

As described, we have kept hyperparameters as similar as possible to the original GaussTR to enable fair comparison. However, since we have made major modifications to the model, it is not certain that the hyperparameters used originally yields optimal performance in our case. Inspired by GaussianFormer [15] and GaussianFlowOcc [3], which both use a considerably higher number of Gaussians, we also try increasing the number of Gaussians in our model. When using more Gaussians, we thought it would be reasonable to try decreasing the size of each individual Gaussian, allowing for a more detailed representation. We got the best IoU results when increasing the total number of Gaussians from 1800 to 2400 while decreasing the maximum scale of Gaussians by 37.5%. Results from three such runs are presented in table 7.

Method	**IoU**
No LiDAR supervision + original ResNet	37.17
LiDAR supervision + original ResNet	36.10
LiDAR supervision + distilled ResNet	41.00

Table 7: Occupancy prediction results on the Occ3D-nuScenes validation set using 2400 Gassians with 37.5% smaller max scale.

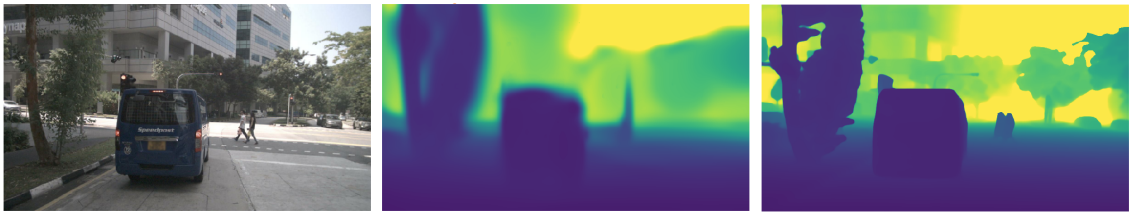
The "No LiDAR supervision + original ResNet" configuration is directly comparable to "Base" from Table 6, differing only in the size and scale of the Gaussians. Similarly, "LiDAR supervision + original ResNet" and "LiDAR supervision + distilled ResNet" are directly comparable to "Base L" and "Base D+L", respectively. We find that these changes increase the IoU from 34.63 to 37.17, from 34.36 to 36.10, and from 39.52 to 41.00, respectively. Corresponding depth maps for these configurations can be seen in Figures 11, 12, and 13. We note that all evaluated configurations show improved performance with these modifications. It is possible that continuing to change these parameters, or other hyperparameters, would yield even higher performance.



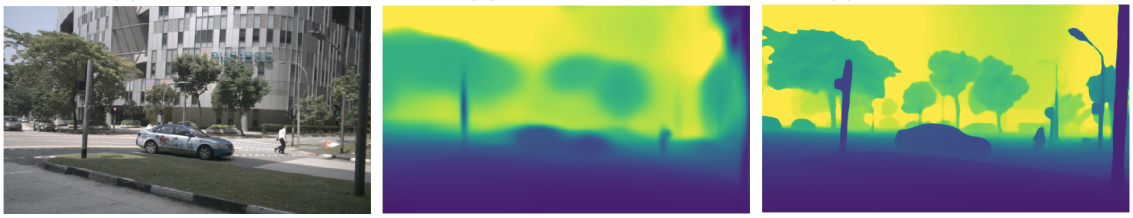
(a) RGB image **(b)** Rendered depth **(c)** Metric3D GT depth

(d) RGB image **(e)** Rendered depth **(f)** Metric3D GT depth

Figure 11: Comparison between rendered depth maps and Metric3D pseudo-ground-truth depth maps from our model with **original ResNet** and **no LiDAR**, together with the corresponding RGB images. The upper row corresponds to the front camera, while the lower row corresponds to the front-right camera. Both rows are taken from the same frame.



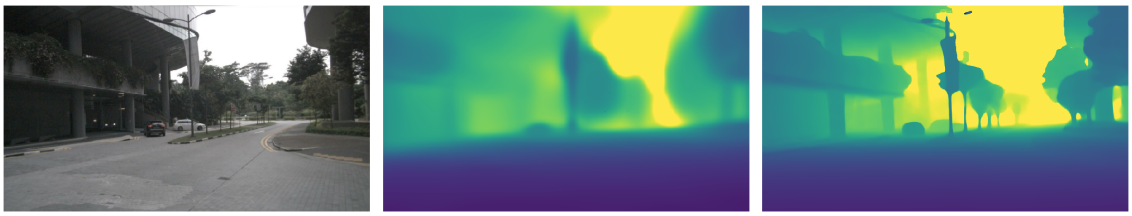
(a) RGB image **(b)** Rendered depth **(c)** Metric3D GT depth



(d) RGB image **(e)** Rendered depth **(f)** Metric3D GT depth

Figure 12: Comparison between rendered depth maps and Metric3D pseudo-ground-truth depth maps from our model with **original ResNet** and **LiDAR**, together with the corresponding RGB images. The upper row corresponds to the front camera, while the lower row corresponds to the front-right camera. Both rows are taken from the same frame.

(a) RGB image **(b)** Rendered depth **(c)** Metric3D GT depth



(d) RGB image **(e)** Rendered depth **(f)** Metric3D GT depth

Figure 13: Comparison between rendered depth maps and Metric3D pseudo-ground-truth depth maps from our model with **distilled ResNet** and **LiDAR**, together with the corresponding RGB images. The upper row corresponds to the front camera, while the lower row corresponds to the front-right camera. Both rows are taken from the same frame.

Interesting to note is that both rendered depth maps that utilize LiDAR, figure 12 and 13, are more detailed compared to the rendered depth map in figure 11, which does not use LiDAR. However, despite this, ”No LiDAR supervision + original

ResNet" manages to achieve higher IoU than "LiDAR supervision + original ResNet", in accordance with the results from the previous section.

5.7 Semantic Tuning

In this section we present results of our semantic tuning. Since we are only interested in the quality of features in this section we will base the semantic tuning on the pre-trained checkpoints with the highest IoU score, while still being trained with self-supervision. Since it directly impacts feature quality, we will present results with both the original and distilled ResNet. We also present results where the whole pre-trained model is frozen and when the Gaussian feature head is learnable along with the semantic head. We do not unfreeze any more modules in semantic fine-tuning, as that would affect the geometry of the model, not just semantics.

Method	**IoU**	**mIoU**	barrier	bicycle	bus	car	cons. veh.	motorcycle	pedestrian	traffic cone	trailer	truck	drive. surf.	sidewalk	terrain	manmade	vegetation	others	other flat
No LiDAR U	37.17	4.98	0.00	0.00	0.00	6.26	0.00	0.00	0.08	0.00	0.04	0.98	35.40	6.96	16.26	11.94	6.83	0.00	0.00
Base D	41.00	7.51	0.05	0.00	0.98	14.50	0.00	0.00	0.44	0.00	2.01	5.63	47.43	9.17	16.71	16.23	14.41	0.00	0.02
Base U	36.10	10.03	5.95	0.06	5.08	14.83	1.99	1.14	3.18	1.36	4.18	5.65	47.45	19.49	24.83	15.02	14.49	1.26	4.59
Base D+U	41.00	13.57	12.61	0.94	9.93	19.32	4.62	2.10	3.97	1.52	8.42	10.08	53.49	24.62	31.30	18.21	18.11	1.96	9.42

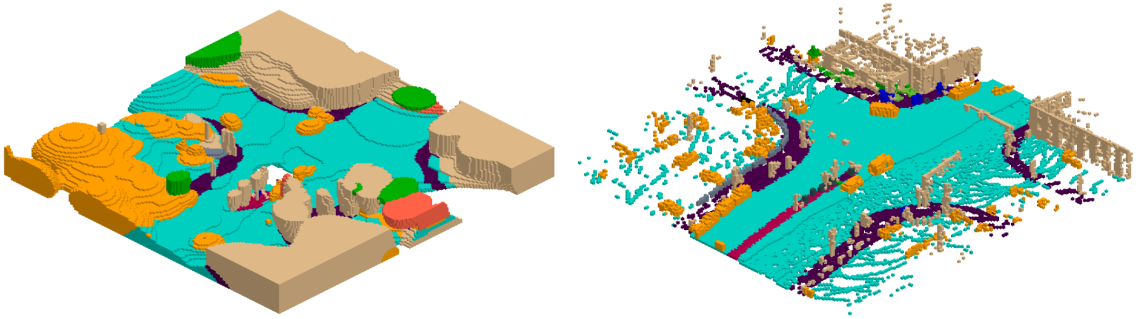
Table 8: Occupancy prediction results on the Occ3D-nuScenes validation set using frozen pre-trained backbone and semantic tuning. Versions with distilled ResNet marked with D and versions unfrozen feature head marked with U. "Base" versions are pre-trained with LiDAR depth supervision.

"Base D+U" in table 8 shows an mIoU of 13.57, which outperforms both our reimplementation and the reported results of GaussTR, as seen in table 2. This demonstrates clear potential for our model to learn semantics, even with ResNet backbone and supervision. "Base D" performs significantly worse. This shows the importance of allowing the Gaussian feature head to be learnable during tuning, demonstrating that directly tuning a semantic head on frozen Gaussian features is not enough for our purposes.

The "Base U" case performs worse than the "Base D+U" version, indicating that distilled ResNet is beneficial for mIoU as well as IoU. "No LiDAR U" performs significantly worse on mIoU than the corresponding "Base U" case, which was pre-trained with LiDAR depth supervision, despite having higher IoU. This suggests that LiDAR, while not always helping IoU, is a strong help for semantic classification.

Unsurprisingly, our semantic tuning yields the best IoU for classes which are commonly occurring in the data, as seen in table 1.

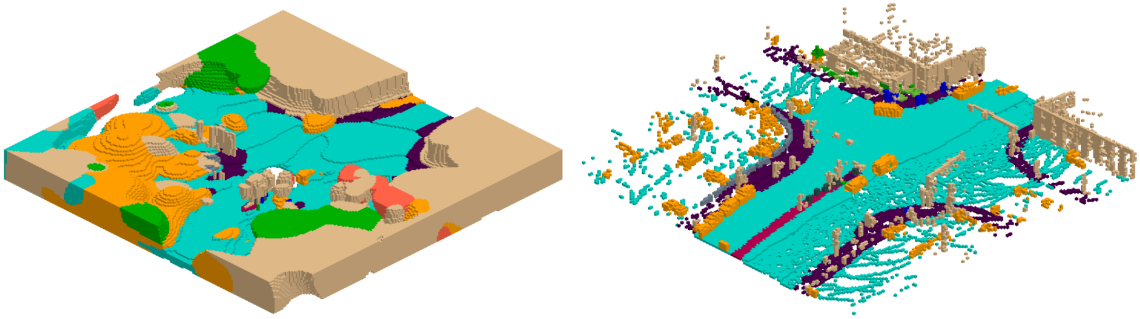
5.7.1 Occupancy Visualizations



(a) Occupancy prediction **(b)** Occ3D-nuScenes GT occupancy

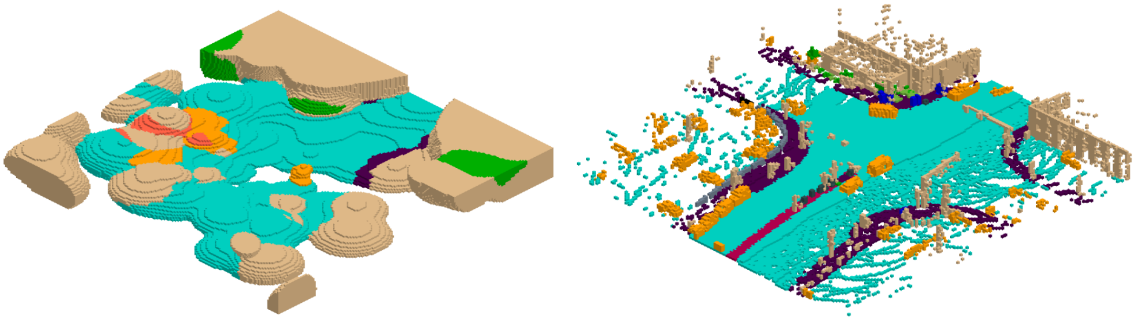
Figure 14: Visualization of occupancy prediction from semantic tuning, with **learnable feature head** of our model with **distilled ResNet** backbone and **LiDAR supervision** and corresponding occupancy GT generated by Occ3D-nuScenes.

Figure 14 shows the occupancy prediction of the "Base D+U" configuration from table 8, with 41.00 IoU and 13.57 mIoU. These numbers show worse binary occupancy but better semantics than the original GaussTR. When comparing figures 10 and 14, it is clear that our method results in more coherent semantic predictions.



(a) Occupancy prediction **(b)** Occ3D-nuScenes GT occupancy

Figure 15: Visualization of occupancy prediction from semantic tuning, with **learnable feature head** of our model with **original ResNet** backbone and **LiDAR supervision** and corresponding occupancy GT generated by Occ3D-nuScenes.



(a) Occupancy prediction **(b)** Occ3D-nuScenes GT occupancy

Figure 16: Visualization of occupancy prediction from semantic tuning, with **learnable feature head** of our model with **original ResNet** backbone and **NO LiDAR supervision** and corresponding occupancy GT generated by Occ3D-nuScenes.

The visualization in figure 16 looks worse than its counterpart which was pre-trained with LiDAR supervision, shown in figure 15, despite having slightly higher IoU. The version with no LiDAR supervision has significantly lower mIoU, which can explain the poor looking visualization.

5.8 Comparison with Other Relevant Works

Here, we present the result of other comparable works, on both IoU and mIoU.

Method	**IoU**	**mIoU**	barrier	bicycle	bus	car	cons. veh.	motorcycle	pedestrian	traffic cone	trailer	truck	drive. surf.	sidewalk	terrain	manmade	vegetation	others	other flat
SelfOcc [14]	45.01	9.30	0.15	0.66	5.46	12.54	0.00	0.80	2.10	0.00	0.00	8.25	55.49	26.30	26.54	14.22	5.60	0.00	0.00
OccNeRF [27]	22.81	9.53	0.83	0.82	5.13	12.49	3.50	0.23	3.10	1.84	0.52	3.90	52.62	20.81	24.75	18.45	13.19	0.00	0.00
DistillNeRF [24]	29.11	8.93	1.35	2.08	10.21	10.09	2.56	1.98	5.54	4.62	1.43	7.90	43.02	16.86	15.02	14.06	15.06	0.00	0.00
GaussianFlowOcc [3]	46.91	17.08	6.75	9.68	18.98	17.15	4.19	11.78	9.27	10.30	1.83	12.33	61.03	31.17	34.78	14.66	12.40	0.00	0.00
GaussTR [16]	45.19	11.70	2.09	5.22	14.07	20.43	5.70	7.08	5.12	3.93	0.92	13.36	39.44	15.68	22.89	21.17	21.87	0.00	0.00
Our Best	**41.00**	**13.57**	12.61	0.94	9.93	19.32	4.62	2.10	3.97	1.52	8.42	10.08	53.49	24.62	31.30	18.21	18.11	1.96	9.42

Table 9: Quantative performance of comparable methods on the Occ3D-nuScenes [22] dataset. The GaussTR result is from their version which includes a segmentation loss, introducing another VFM. This is not included in the version we use as our baseline, presented above.

These works can differ significantly from one another, but they are all evaluated on occupancy prediction and semantic occupancy prediction. SelfOcc [14] can be considered a self-supervised approach, but it only provides an implicit representation of the scene. OccNeRF [27] and DistillNeRF [24] are also based on implicit representations, while additionally relying on pre-trained semantic models or vision foundation models (VFMs). In contrast, GaussianFlowOcc [3] uses an explicit Gaussian-based

scene representation, but also incorporates VFMs. The same is true for GaussTR [16], which was described previously. As shown in table 9, the different methods excel in different areas. Nevertheless, our self-supervised model achieves competitive performance compared to several methods that rely on VFMs or implicit representations, while still maintaining an explicit scene representation.

6 Discussion

6.1 Performance Variation Between Identical Configurations

As mentioned, we used early stopping in all experiments. Evaluation was performed after every training epoch, and the one with highest IoU was presented. However, the IoU-score of the highest performing checkpoint differs, even between runs with identical configurations. This can be due to random initializations and other non-deterministic aspects of training. It was common for IoU to differ with 1 whole percentage point for identical configurations. Results should be interpreted accordingly. A small difference in IoU between two runs might be due to training randomness rather than one configuration actually being meaningfully better.

6.2 mIoU as a Feature-Metric

We use mIoU as our main metric for determining if a model is capable of semantic classification. It can therefore be an indication of how well the model separates visual features. However, mIoU can only be good if the model already has good geometric understanding, measured in our case with binary IoU. It can therefore be misleading to compare the feature quality of two models by mIoU performance. Even with identical feature input and supervision, the model with worse binary IoU will be at a disadvantage. This is important to keep in mind when interpreting our results.

6.3 Semantic Tuning Compared with Zero-Shot Semantics

As seen in our results, our best model achieves better mIoU than the original GaussTR, even with lower binary IoU. This is a strong indication that our training setup is capable of learning meaningful semantics, given that it has good geometric understanding. Additionally, our method has the ability to predict classes ”others” and ”other flat”, which is an obvious advantage compared with GaussTR and other similar methods. However, we lose the possibility for zero-shot open vocabulary semantics. This is unavoidable when not supervising with text-aligned VFM features.

While our best model outperforms GaussTR in mIoU, it is still considerably worse than supervised works like FB-Occ [18] and CTF-Occ [22]. This is no surprise, and would not be a fair comparison, since those models were trained speficially on the task of semantic occupancy prediction. Our model is pre-trained on general scene understanding through Gaussian splatting and photometric consistency warping. It is not optimized for occupancy prediction or semantics during pre-training. We treat these as downstream tasks to show that our pre-training is capable of learning a scene representation which can generalize to many tasks.

6.4 Improving Performance by Tuning for Occupancy

As described, our semantic tuning only modifies the components to do with semantic features, keeping everything to do with geometry frozen. If one wanted to maximize performance on occupancy prediction specifically, it would be possible to take our pre-trained checkpoint and unfreeze more components, including those which affect geometry, and optimize for occupancy prediction. This would likely improve both IoU and mIoU. The benefit of using our pre-trained checkpoint, which was trained in a self-supervised manner, would be that this would likely reduce the amount of labelled data needed compared to other fully supervised methods.

6.5 The Relationship Between Depth and Occupancy Prediction

An interesting observation from the results presented in table 6 and table 7 is that the methods without LiDAR supervision achieved a higher IoU score than the corresponding methods using LiDAR supervision when the original ResNet backbone was used. This trend was, however, not observed when using the distilled ResNet backbone.

One might expect this difference in occupancy performance to also be reflected in the quality of the rendered depth maps, or at least result in depth maps of similar quality for both approaches. However, this is not what we observe. In Section 5.6, where the rendered depth maps are visualized, the depth map shown in figure 12 contains noticeably more detail and structure compared to the rendered depth map in figure 11.

The reason for this behavior is not entirely clear, and it appears that the relationship between detailed rendered depth maps and strong occupancy prediction performance is more complex than one might initially expect. Ideally, an accurate rendered depth map should originate from Gaussians that are correctly placed at the true depth for each pixel. In such a case, these Gaussians would also contribute to occupancy in the correct voxels in 3D space. From the results, we observe that all

models using Metric3D depth during inference achieve high IoU scores. Since the Gaussians in these cases are lifted using the estimated "true" depth, the model tends to produce strong occupancy predictions. Similarly, table 5 shows that supervising the model using the "true" depth map results in high IoU scores even when using the original ResNet backbone.

The key observation is that a visually accurate rendered depth map does not necessarily imply that the Gaussians are correctly positioned in 3D space. Since the rendered depth for a pixel is obtained by aggregating contributions from multiple Gaussians along the viewing ray, multiple Gaussian configurations may produce similar depth maps. For some reason, when combining LiDAR supervision with the original ResNet backbone, the model is able to generate detailed rendered depth maps without a corresponding improvement in occupancy prediction performance. This suggests that some Gaussians may not be correctly placed in 3D space, even though they are positioned in a way that allows the rendered depth map to better match the target depth map from the LiDAR supervision. A conceptual illustration of this phenomenon is shown in figure 17, where two incorrectly positioned Gaussians can still produce a rendered depth that satisfies the LiDAR supervision signal.

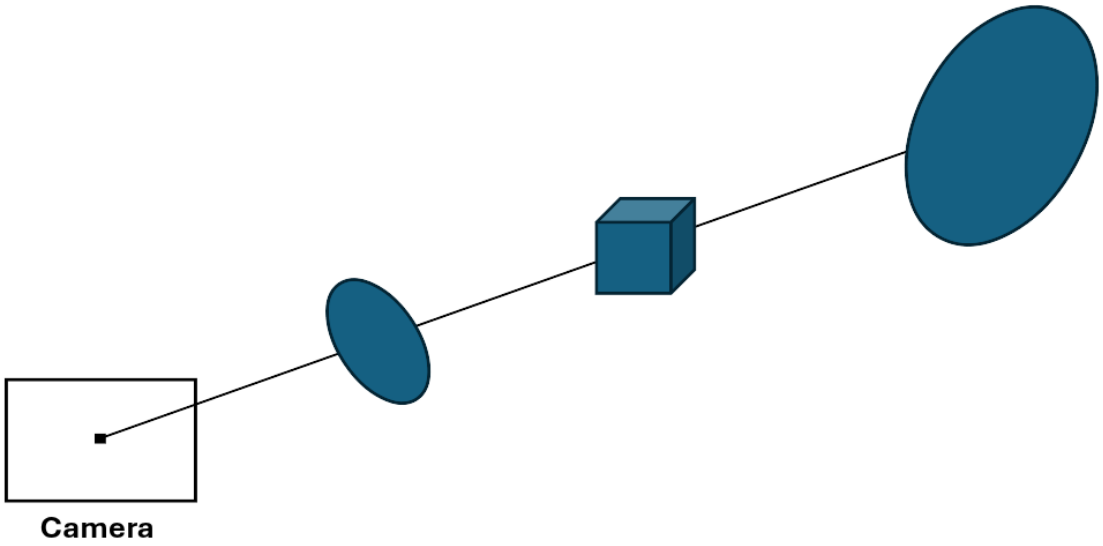


Figure 17: Illustration of how Gaussians (blue ovals) positioned along a viewing ray can contribute to a "correct" rendered depth for a specific pixel, while still not contributing to the true occupied voxel (blue cube) where the Gaussians ideally should be placed.

There are several interesting aspects to discuss here. In table 5, we achieve high IoU scores even when using the original ResNet backbone. One possible explanation

is that supervision using Metric3D provides a significantly stronger depth signal compared to LiDAR supervision, since the Metric3D depth maps are considerably denser. When supervising using LiDAR depth, the model only compares the rendered depth map against a sparse set of LiDAR depth values. As a result, the supervision does not directly enforce that individual Gaussians are placed at the correct depth in 3D space. This may allow the model to learn Gaussian configurations that produce visually accurate rendered depth maps without necessarily improving the underlying occupancy prediction.

One possible direction for improving this behavior would be to supervise the depth of individual Gaussians directly, instead of only supervising the final rendered depth map. For example, the depth of each Gaussian contributing to a pixel could be compared against the corresponding LiDAR depth value for that pixel. Such an approach could potentially encourage the Gaussians to move toward more accurate 3D positions, while still maintaining detailed rendered depth maps. In this case, the LiDAR supervision would operate at the Gaussian level rather than only comparing rendered depth against LiDAR depth.

The most surprising observation, however, is that this behavior is not present in table 6 when using the distilled ResNet backbone. In that case, the method using LiDAR supervision achieves significantly better occupancy prediction performance compared to the corresponding method without LiDAR supervision. This might suggest that the relationship between rendered depth quality and occupancy prediction performance depends on the underlying feature representation used by the model. With the distilled ResNet backbone, the model is able to generate detailed rendered depth maps while also being able to generate more "correct" geometry, shown through the high IoU score. Since the distilled ResNet has inherited information from the richer FeatUp representations through the distillation process, this indicates that the quality and semantic structure of the learned features may play an important role in how effectively the model can utilize our LiDAR supervision.

6.6 Effect of Warping Loss

From the results in figure 11, it is clear that photometric consistency loss, along with splatted feature loss, can produce reasonable large-scale depth. However, we struggle more with fine details in the depth map than Monodepth2 [10] when not using LiDAR supervision. As mentioned, we did not find that edge-aware smoothing improved our results. From one perspective it should be easier for us to learn depth than for Monodepth2, since we have pose for all images, while Monodepth2 needs to learn pose along with depth. However, it is clear that we do not learn depth as effectively. There can be a few possible reasons for this.

Firstly we need to learn features for each Gaussian along with depth, which may

distract from pure depth learning. Second, we use an explicit Gaussian representation, where depth maps are generated from Gaussian splatting. Monodepth2 produces depth maps directly. There are many possible Gaussian configurations which result in the same rendered depth in a pixel. As discussed above, this may result in correct rendered depth, but wrong 3D representation. Additionally, it may make it harder for the model to learn correct rendered depth in the first place. To combat these difficulties, it would be beneficial with some sort of supervision directly in 3D Gaussian space, if it can be done in a self-supervised manner. Another option might be to predict a dense depth map directly and use that depth map for photometric consistency warping, as well as to place Gaussians in 3D space. Currently, we only predict depth directly for reference points.

6.7 VFM and LiDAR Dependence

Our results show that strong occupancy performance can still be achieved without using large VFMs such as MaskCLIP-FeatUp or Metric3D at inference time. This is significant from a practical perspective, since removing VFMs at inference reduces computational cost, memory usage, and latency, while simplifying deployment in real-world autonomous driving systems. Although using VFMs as supervision signals still provides valuable semantic and geometric information, our experiments demonstrate that much of this information can be transferred into a considerably lighter ResNet-50 backbone through distillation. Compared to relying directly on a large ViT-based VFM, dependence on a standard ResNet, pre-trained on ImageNet [6], is less restrictive, more computationally efficient, and easier to reproduce and deploy.

Our fully self-supervised experiments further show that meaningful geometric and semantic scene understanding can emerge directly from multi-view consistency, Gaussian scene representations, and sparse LiDAR supervision, even without external semantic supervision. Results show that the self-supervised model is capable of learning coarse geometric structure without LiDAR supervision, while LiDAR supervision helps the model learn finer geometric details, particularly visible in the rendered depth maps. Versions using LiDAR supervision also perform better during semantic fine-tuning, which suggests that the additional geometric detail provided by the LiDAR signal is beneficial for downstream semantic classification. While the fully self-supervised models do not match the performance of methods relying heavily on VFMs or task-specific supervision, they still achieve competitive occupancy results. Interestingly, the distilled ResNet backbone consistently improves performance compared to the original ResNet, particularly when combined with LiDAR supervision. This suggests that stronger feature representations help the model utilize sparse geometric supervision more effectively. Overall, the results indicate that large VFMs are not strictly necessary for learning useful 3D scene representations, and that sev-

eral of their advantages can be retained through lightweight distilled representations together with self-supervised geometric learning.

7 Future Work

7.1 Downstream Tasks and Temporal Prediction

Our model has the ability, through self-supervised training, to create an explicit 3D representation of the surrounding scene around the vehicle, while also maintaining semantic separation between different objects and regions in the environment. We have shown that this enables the model to achieve strong performance on occupancy prediction, as well as performing well when further trained on semantic occupancy prediction as a downstream task. This leaves room for exploring additional downstream tasks that the pre-trained model could potentially be optimized for.

The demonstrated ability to perform semantic occupancy prediction suggests that the model could potentially be further trained for object detection. While semantic occupancy prediction focuses on predicting the semantic class of individual voxels, object detection instead requires the network to reason about which regions of voxels belong to the same physical object. This could allow the model to distinguish between closely placed objects of the same semantic class, such as multiple nearby vehicles or pedestrians. Since the learned Gaussian representation already contains both semantic and geometric information about the scene, it is possible that the representation could serve as a strong foundation for object-level reasoning tasks.

Another interesting direction for future work would be to introduce a temporal component to the model, inspired by recent work on temporal occupancy prediction and world models such as SparseWorld [5]. At the moment, the network only predicts Gaussians corresponding to the current input images. Extending the model to also predict future Gaussian representations across multiple time steps could enable the network to reason about how objects and regions in the scene move over time. Such temporal understanding could potentially improve the consistency of the scene representation while also enabling motion-related downstream tasks. If combined with object detection capabilities, this may further allow the model to estimate and predict the future movement of detected objects, such as pedestrians walking or vehicles moving through the scene.

7.2 Generalization to Other Datasets

So far, the model has only been trained and evaluated on the nuScenes dataset. It would therefore be interesting to investigate how well the proposed approach gener-

alizes to other datasets and environments. Since nuScenes contains data collected from a limited set of urban environments and sensor setups, there is a risk that the model adapts too strongly to the specific characteristics of the dataset. Evaluating the model on additional datasets with different environments, road structures, weather conditions, and sensor configurations could therefore provide valuable insight into the robustness and transferability of the learned scene representation.

One possible dataset for such experiments is Occ3D-Waymo [22], which is based on the Waymo Open Dataset [21] and provides occupancy annotations similar to Occ3D-nuScenes. Compared to nuScenes, the Waymo dataset contains different driving environments and sensor characteristics, making it a relevant benchmark for evaluating cross-dataset generalization and the adaptability of the proposed approach.

7.3 Multimodal Input and Enhanced Use of LiDAR

While we achieved promising results on both pre-training and our chosen downstream task, there are still several aspects of the project that could be further explored.

At the moment, the provided LiDAR data is only used to help the model learn accurate depth through the loss function. One direction that has not been explored in this project is the use of multimodal input. By instead using the LiDAR sweeps directly as input data to the network, it may be possible to encode information about occupied regions directly into the Gaussian queries. Another possible direction would be to use the LiDAR depth map at the stage where the Gaussian queries are lifted into 3D by the MLPs in the Gaussian head. By using the measured LiDAR depth instead of the predicted depth for pixels where LiDAR depth values are available, some Gaussians could potentially be lifted more accurately into 3D space.

It may also be possible to further improve the current use of the LiDAR depth map. Due to the sparse nature of LiDAR sweeps compared to the number of pixels in an image, only a relatively small subset of pixels are assigned valid depth values. One possible approach to mitigate this issue would be to interpolate depth values between nearby pixels with valid LiDAR depth measurements in order to create a denser depth map. For example, if several pixels belonging to the same object contain LiDAR depth measurements while nearby pixels do not, interpolating depth values between them could potentially improve the effect of the loss. Since many objects often have relatively consistent depth across neighboring pixels, this may allow the model to learn more accurate geometric representations.

Another possible extension would be to explore the use of RADAR data as an additional sensor modality. Since RADAR data is already provided in the nuScenes dataset [4], it would be a natural direction to investigate if multimodal input were to be explored further. Compared to cameras and LiDAR, RADAR also offers different sensing characteristics and may provide complementary information about the scene,

particularly in challenging environments or for moving objects. While RADAR data is generally sparse and noisy, combining it with camera and LiDAR information could potentially improve the robustness of the learned scene representation.

One caveat with multimodal approaches is that they require additional sensors during inference. If the model were to be deployed in a vehicle using LiDAR or RADAR as input modalities, the vehicle would also need to be equipped with these sensors. This increases both system complexity and deployment cost, while also introducing additional sources of sensor noise and calibration challenges. One advantage of the proposed approach is that LiDAR is currently only used during training, allowing inference to be performed using camera input alone.

7.4 Different Backbone

The distilled ResNet performs significantly better than the original ResNet in our experiments. This suggests that the quality of features from the backbone is important. Ideally, we do not want to use the distilled version, since it contains a VFM-dependance. Therefore it would be a suitable next step to try another backbone, which has no such VFM-dependence, but still produces higher quality features than the original ResNet. One such backbone to try could be ConvNext [19], which can be viewed as a "modernized" version of ResNet.

8 Conclusion

The two important aspects of a vision-based world model, which we focus on, are geometry and features. Achieving good geometric understanding and feature representations is made easier by the use of VFMs. However, this introduces dependencies on complex external models, which we have aimed to get rid of. Our results show that it is possible to learn a meaningful model in a fully self-supervised manner based on an explicit Gaussian representation, even though performance decreases slightly compared with our VFM-assisted baseline. When using our pre-trained model for a chosen downstream task, semantic occupancy prediction, we achieve competitive results. Overall our results indicate that geometry and features are not totally decoupled. A good geometric understanding facilitates feature separation, while good feature separation makes it easier to learn correct geometry.

9 References

[1] Ben Agro, Quinlan Sykora, Sergio Casas, Thomas Gilles, and Raquel Urtasun. Uno: Unsupervised occupancy fields for perception and forecasting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 14487–14496. IEEE, 2024. doi: 10.1109/CVPR52733.2024.01373. URL https://doi.org/10.1109/CVPR52733.2024.01373.

[2] Maxim Berman, Amal Rannen Triki, and Matthew B. Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4413–4421. Computer Vision Foundation / IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00464. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Berman_The_LovaSz-Softmax_Loss_CVPR_2018_paper.html.

[3] Simon Boeder, Fabian Gigengack, and Benjamin Risse. Gaussianflowocc: Sparse and weakly supervised occupancy estimation using gaussian splatting and temporal flow. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 24943–24954, October 2025.

[4] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 11618–11628. Computer Vision Foundation / IEEE, 2020. doi: 10.1109/CVPR42600.2020.01164. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Caesar_nuScenes_A_Multimodal_Dataset_for_Autonomous_Driving_CVPR_2020_paper.html.

[5] Chenxu Dang, Haiyan Liu, Jason Bao, Pei An, Xinyue Tang, An Pan, Jie Ma, Bingchuan Sun, and Yan Wang. Sparseworld: A flexible, adaptive, and efficient 4d occupancy world model powered by sparse and dynamic queries. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor, editors, *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pages 3497–3505. AAAI Press, 2026. doi: 10.1609/AAAI.V40I5.37347. URL https://doi.org/10.1609/aaai.v40i5.37347.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPR.2009.5206848. URL https://doi.org/10.1109/CVPR.2009.5206848.

[7] Xiaoyi Dong, Jianmin Bao, Yinglin Zheng, Ting Zhang, Dongdong Chen, Hao Yang, Ming Zeng, Weiming Zhang, Lu Yuan, Dong Chen, Fang Wen, and Nenghai Yu. Maskclip: Masked self-distillation advances contrastive language-image pretraining. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 10995–11005. IEEE, 2023. doi: 10.1109/CVPR52729.2023.01058. URL https://doi.org/10.1109/CVPR52729.2023.01058.

[8] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2366–2374, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/7bccfde7714a1ebadf06c5f4cea752c1-Abstract.html.

[9] Stephanie Fu, Mark Hamilton, Laura E. Brandt, Axel Feldmann, Zhoutong Zhang, and William T. Freeman. Featup: A model-agnostic framework for features at any resolution. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=GkJiNn2QDF.

[10] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow. Digging into self-supervised monocular depth estimation. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 3827–3837. IEEE, 2019. doi: 10.1109/ICCV.2019.00393. URL https://doi.org/10.1109/ICCV.2019.00393.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

[12] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

[13] Mu Hu, Wei Yin, Chi Zhang, Zhipeng Cai, Xiaoxiao Long, Hao Chen, Kaixuan Wang, Gang Yu, Chunhua Shen, and Shaojie Shen. Metric3d v2: A versatile monocular geometric foundation model for zero-shot metric depth and surface normal estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[14] Yuanhui Huang, Wenzhao Zheng, Borui Zhang, Jie Zhou, and Jiwen Lu. Selfocc: Self-supervised vision-based 3d occupancy prediction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 19946–19956. IEEE, 2024. doi: 10.1109/CVPR52733.2024.01885. URL https://doi.org/10.1109/CVPR52733.2024.01885.

[15] Yuanhui Huang, Wenzhao Zheng, Yunpeng Zhang, Jie Zhou, and Jiwen Lu. Gaussianformer: Scene as gaussians for vision-based 3d semantic occupancy prediction. In Ales Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part XXVII*, Lecture Notes in Computer Science, pages 376–393. Springer, 2024. doi: 10.1007/978-3-031-73383-3_22. URL https://doi.org/10.1007/978-3-031-73383-3_22.

[16] Haoyi Jiang, Liu Liu, Tianheng Cheng, Xinjie Wang, Tianwei Lin, Zhizhong Su, Wenyu Liu, and Xinggang Wang. Gausstr: Foundation model-aligned gaussian transformer for self-supervised 3d spatial understanding. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025, Nashville, TN, USA, June 11-15, 2025*, pages 11960–11970. Computer Vision Foundation / IEEE, 2025. doi: 10.1109/CVPR52734.2025.01117. URL https://openaccess.thecvf.com/content/CVPR2025/html/Jiang_GaussTR_Foundation_Model-Aligned_Gaussian_Transformer_for_Self-Supervised_3D_Spatial_Understanding_CVPR_2025_paper.html.

[17] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139:1–139:14, 2023. doi: 10.1145/3592433. URL https://doi.org/10.1145/3592433.

[18] Zhiqi Li, Zhiding Yu, David Austin, Mingsheng Fang, Shiyi Lan, Jan Kautz, and Jose M. Alvarez. FB-OCC: 3D occupancy prediction based on forward-backward

view transformation. In *CVPR Workshop on End-to-end Autonomous Driving*, June 2023.

[19] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 11966–11976. IEEE, 2022. doi: 10.1109/CVPR52688.2022.01167. URL https://doi.org/10.1109/CVPR52688.2022.01167.

[20] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, Proceedings of Machine Learning Research, pages 8748–8763. PMLR, 2021. URL http://proceedings.mlr.press/v139/radford21a.html.

[21] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[22] Xiaoyu Tian, Tao Jiang, Longfei Yun, Yucheng Mao, Huitong Yang, Yue Wang, Yilun Wang, and Hang Zhao. Occ3d: A large-scale 3d occupancy prediction benchmark for autonomous driving. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/cabfaeecaae7d6540ee797a66f0130b0-Abstract-Datasets_and_Benchmarks.html.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural*

Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[24] Letian Wang, Seung Wook Kim, Jiawei Yang, Cunjun Yu, Boris Ivanovic, Steven L. Waslander, Yue Wang, Sanja Fidler, Marco Pavone, and Péter Karkus. Distillnerf: Perceiving 3d scenes from single-glance images by distilling neural fields and foundation model features. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/720991812855c99df50bc8b36966cd81-Abstract-Conference.html.

[25] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861. URL https://doi.org/10.1109/TIP.2003.819861.

[26] Vickie Ye, Ruilong Li, Justin Kerr, Matias Turkulainen, Brent Yi, Zhuoyang Pan, Otto Seiskari, Jianbo Ye, Jeffrey Hu, Matthew Tancik, and Angjoo Kanazawa. gsplat: An open-source library for gaussian splatting. *J. Mach. Learn. Res.*, 26: 34:1–34:17, 2025. URL https://jmlr.org/papers/v26/24-1476.html.

[27] Chubin Zhang, Juncheng Yan, Yi Wei, Jiaxin Li, Li Liu, Yansong Tang, Yueqi Duan, and Jiwen Lu. Occnerf: Advancing 3d occupancy prediction in lidar-free environments. *IEEE Trans. Image Process.*, 34:3096–3107, 2025. doi: 10.1109/TIP.2025.3567828. URL https://doi.org/10.1109/TIP.2025.3567828.

[28] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: deformable transformers for end-to-end object detection. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=gZ9hCDWe6ke.

Master's Theses in Mathematical Sciences 2026:E32
ISSN 1404-6342

LUTFMA-3618-2026

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/