UNIVERSITY OF PISA

DEPARTMENT OF COMPUTER SCIENCE

Master Degree in Computer Science

Semantic Querying of Reconstructed 3D Environments Using Pre-trained 2D Foundation Models

Supervisors:

Daniela Giorgi

Fabio Carrara

Gianpaolo Palma

Candidate:

Paolo Fasano

ACADEMIC YEAR 2023/2024

Abstract

Understanding and interacting with complex 3D environments is increasingly important in robotics, virtual reality, and autonomous systems. To address this need, this thesis project aims to develop a system that reconstructs a 3D model of the environment enhanced with semantic features. We project in the 3D model the semantic features extracted using pre-trained 2D foundational models, such as Segment Anything and OpenCLIP. This allows open-vocabulary queries about the environment in natural language that are not restricted by predefined categories or information and without requiring additional training. The system is validated on real RGB-D data captured from a HoloLens 2 device.

Contents

Chapter 1 Introduction

The rapid advancements in fields such as augmented reality (AR), robotics, virtual reality (VR), autonomous systems, and artificial intelligence (AI) have underscored the growing importance of comprehending, analyzing, and interacting with complex three-dimensional (3D) environments. These technologies increasingly rely on accurate and semantically rich representations of the physical world to enable more efficient decision-making, navigation, and interaction.

In response to the growing demand for systems capable of understanding and interacting with complex 3D environments, this thesis focuses on the design and implementation of a system that reconstructs a 3D model, also known as a mesh, of the environment augmented with semantic features. The primary aim of this system is to generate not only a geometrically accurate representation of the surroundings but also to embed meaningful contextual information, thereby enhancing the interpretability and usability of the reconstructed model. By integrating semantic understanding into the 3D reconstruction process, this work seeks to bridge the gap between raw spatial data and its application in advanced technological domains, facilitating the development of more intelligent and context-aware systems. In other words we want to query a reconstructed scene asking something thru the use of natural language, for instance we may want to ask "Where can i sit?" and expect the system to yeld the portions of the reconstructed mesh that answer such query.

To reach the goal of making such system, we leveraged foundational models to extract features and enable zero-shot querying. Foundational models, such as large-scale vision and language models, have been trained on extensive and diverse datasets, resulting in highly generalized representations capable of addressing a wide variety of tasks. Developing new, task-specific models from scratch is often prohibitively expensive, requiring significant computational resources, large amounts of labeled data, and extensive expertise. In contrast, foundational models provide a cost-effective and efficient alternative by offering pre-trained capabilities that can be utilized for diverse applications without the need for additional fine-tuning. This work capitalizes on the zero-shot capabilities of foundational models, enabling the system to address complex queries without requiring further training. By doing so, the approach not only reduces computational and developmental overhead but also ensures that the system remains flexible and adaptable to a wide range of use cases. This strategy underscores the practical advantages of harnessing the power of existing foundational models, allowing for efficient deployment while avoiding the challenges associated with bespoke model creation.

The pipeline that will be presented in the following thesis is composed as in Figure 1

Chapter 2, **Background**, introduces the foundational technologies employed in this work, covering both software and hardware aspects. It begins with an explanation of machine learning concepts, tracing the progression from simple artificial neurons to the sophisticated models used today. This chapter also details the foundational models utilized in the development of the system, emphasizing their role in achieving

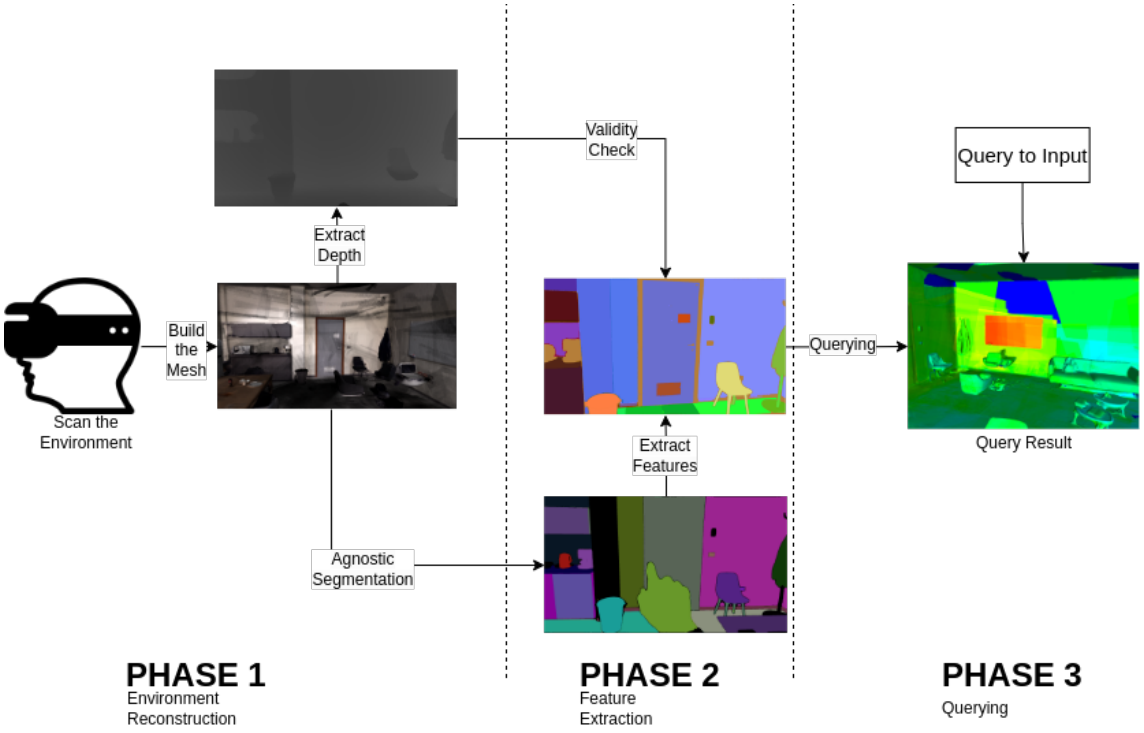


Figure 1: Pipeline of our approach to the task

the thesis objectives. Additionally, it outlines the tools and processes that enable the transformation of a physical environment into a digital 3D mesh, providing the groundwork for subsequent steps.
Chapter 3, **Related Works**, reviews existing models and systems relevant to the thesis objectives. It examines approaches that align with or influence the goals of this work, highlighting the strengths and limitations of current state-of-the-art methods. This review serves to contextualize the contributions of this thesis within the broader research landscape.
Chapter 4, **Method**, presents a detailed account of the proposed system, delineating the phases and steps required to process raw data and perform querying over the extracted features. This chapter introduces the two distinct approaches implemented for feature extraction and querying, offering a thorough explanation of the methodologies and their underlying principles.
Chapter 5, **Experiments and Results**, demonstrates the effectiveness of the designed system through experimental evaluations. It includes a systematic presentation of results, showcasing the system's capabilities and its potential to address the outlined objectives. The chapter concludes with a discussion of limitations and failure analysis, identifying areas where the approach falls short of the state-of-the-art.
Chapter 6, **Conclusions**, presents the final consideration over this thesis project as well as suggestions aimed for mitigating the issues underlined in the failure analysis for future improvements to enhance the system's robustness and performance.

In summary, the objective of this work is to develop a system capable of integrating semantic understanding into a 3D reconstruction mesh generated from real-world

data, which can be queried, in a zero-shot manner, using natural language. Zero-shot querying means that our system is capable of answering the queries over the 3D environment without the need for further training of the AI models. This system was implemented by integrating foundational models and processing their outputs, ultimately achieving promising results, as demonstrated in the following chapters.

Chapter 2 Background

The following section will be used to briefly introduce a set of technologies, hardware, and software that have been used in this thesis, as well as, the theory behind some of the models used in the development. The main hardware component we had to use was the HoloLens 2, which is a mixed-reality headset developed by Microsoft. On the software side of things we used some foundational models, such as Open-Clip [18], Segment Anything [14], DinoV2 [1]; the Visualization and Computer Graphics Library (VCG-Lib) [3] has been used to manage the 3D portion of the project; finally Embree has been used to perform parallelization.

2.1 Machine learning

Machine learning (ML) [17] is a branch of artificial intelligence (AI) that centers on creating computer algorithms capable of improving automatically through experience and data usage. Simply put, machine learning allows computers to learn from data and make decisions or predictions without explicit programming.

In traditional programming, a computer follows a specific set of predefined instructions to complete a task. Conversely, in machine learning, the computer is provided with a dataset and a task to perform, but it must determine how to achieve the task based on the examples it receives. ML models can predict numerical values based on historical data, categorize events as true or false, and cluster data points based on commonalities.

In general, machine learning can be divided into three categories: supervised learning, unsupervised learning, and semi-supervised learning.

Supervised learning is a machine learning approach that is defined by its use of labeled datasets. Labeling a dataset means assigning, at each example in the dataset, the expected resulting value after the operation that we wish the ML algorithm to perform. Using labeled inputs and outputs, the model can measure its accuracy and learn over time. Supervised learning can be separated into two types of problems: classification, where the ML model must devise the input in two or more categories, and regression, where the ML model must understand the relationship between dependent and independent variables.

Unsupervised learning uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms uncover hidden patterns in the data without requiring human interaction. Unsupervised learning can be divided into three problems: clustering, a data mining technique for grouping unlabeled data based on their similarities or differences, association, which uses different rules to find relationships between variables in a given data set, and dimensionality reduction, is a learning technique that is used when the number of features (or dimensions) in a given data set is too high. It reduces the number of data inputs to a manageable size while also preserving the data integrity.

Semi-supervised learning is a balanced approach that utilizes a training dataset containing both labeled and unlabeled data. This method is particularly beneficial when extracting relevant features from the data is challenging and when dealing with a large volume of data.

Deep learning neural networks [17], also known as artificial neural networks, aim to emulate the human brain using a combination of data inputs, weights, and biases. These components collaborate to effectively recognize, classify, and describe objects within the data. Deep neural networks are composed of multiple layers of interconnected nodes. Each layer builds on the previous one to refine and optimize predictions or categorizations. The process of passing computations through the network is known as forward propagation. The input and output layers, referred to as visible layers, mark the beginning and end points of data processing in a deep learning model. The input layer intakes data for processing, while the output layer delivers the final prediction or classification.

The general idea of a Neural Network is to find a model that, based on a set D of N samples, will approximate an unknown function f as well as possible.

$$\begin{aligned} D = \{[x_1, ..., x_N], [y_1, ..., y_N]\} \\ f(x_i) = y_i \end{aligned} \tag{1}$$

where the matrix $[x_1, ..., x_N]$ is the input and $[y_1, ...y_N]$ is the output matrix and $f(x_i)$ is the loss function.

As stated previously, a Neural Network(NN) is always composed of an input layer, an output layer, and, in between, at least a hidden layer, as shown in Figure 2. The layers are connected by weighted transitions, which can be modeled by multiplying a weight matrix with a vector representing the output of the previous layer. Each layer consists of several units called neurons. The number of units in each layer defines its dimension. The activation functions in the neurons effectively model a threshold that determines whether the information received by a neuron is relevant for further calculations. Relevant information is passed on to the next layer until it reaches the output layer. During the training phase, the model is provided with a sample data set D where the matrix $[x_1, ..., x_N]$ is the input and $[y_1, ...y_N]$ is the output matrix. After processing each sample $(x_i, y_i) \in D$, the distance of the output that the network produces to the actual desired output y_i, is measured by some loss function, for example, the L2-norm or the Euclidean norm. Based on the results, the weights of the transitions are adjusted to achieve better performance in the next iteration. The dimension of the input layer is determined by the size of the input data, while the dimension of the output layer corresponds to the specific problem the model is designed to solve.

2.1.1 Fundamental Concepts of Artificial Neural Network

Now that we have a general idea of what machine learning is and how it works, we must introduce some fundamental concepts such as how a single neuron works, what feed-forward and back-propagation are, what the bias is, the activation functions, and the learning.

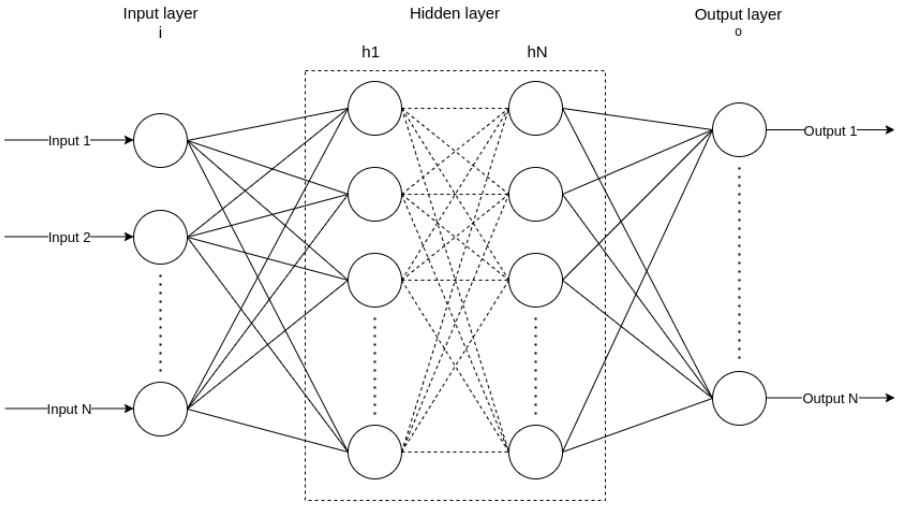


Figure 2: Example of a Neural Network structure

Single Neuron Regardless of the size or purpose of a neural network, the fundamental components are the so-called neurons. A single neuron can be mathematically defined as follows:

$$u_i = \sum_{i=1}^{n} w_i x_i \tag{2}$$

$$y_i = f(u_i + b_i) \tag{3}$$

In these equations, the term u_i represents the input to the i^{th} neuron, which is computed as the weighted sum of all inputs x_i, where w_i denotes the respective weights. The output y_i of the neuron is then determined by applying an activation function f to the sum of u_i and a bias term b_i. The bias b_i is an additional parameter that allows the model to fit the data more effectively by shifting the activation function, thereby improving the learning capacity of the neuron.

Activation functions An activation function determines whether a neuron should be activated based on the input it receives. This means that it decides whether the input to the neuron is significant for the process of making a prediction, by utilizing mathematical operations. The activation function is crucial for deriving the output of a neuron from the input values provided to it. The primary role of an activation function is to introduce non-linearity into the neural network. Without non-linear activation functions, the neural network would be restricted to learning only linear transformations of the input, regardless of the number of layers. This limitation arises because the composition of multiple linear functions remains a linear function. Thus, activation functions enable the network to approximate complex, non-linear relationships between the input and output. The following are some commonly used activation functions in neural networks:

- **Sigmoid Function**: The sigmoid activation function is defined as

$$f(x) = \frac{1}{1 + e^{-x}}\,. \tag{4}$$

This function maps the input to a range between 0 and 1, which is useful for binary classification problems.

- **Tanh Function**: The hyperbolic tangent (tanh) function is defined as

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}\,. \tag{5}$$

The tanh function maps the input to a range between -1 and 1, and is often used when the output needs to be centered around zero.

- **ReLU (Rectified Linear Unit)**: The ReLU activation function is defined as

$$f(x) = \max(0, x)\,. \tag{6}$$

ReLU introduces non-linearity by outputting zero for negative inputs and the input value itself for positive inputs. It is widely used due to its simplicity and effectiveness in mitigating the vanishing gradient problem.

- **Leaky ReLU**: The leaky ReLU activation function is a variation of ReLU and is defined as

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}, \tag{7}$$

where α is a small positive constant. Leaky ReLU allows a small, non-zero gradient for negative inputs, addressing the issue of "dead neurons" in ReLU.

- **Softmax Function**: The softmax activation function is typically used in the output layer for multi-class classification problems and is defined as

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}\,. \tag{8}$$

Softmax normalizes the output to represent a probability distribution across multiple classes.

These activation functions are critical in enabling the neural network to learn and model complex relationships, ensuring that the network is capable of capturing non-linear patterns in the data.

Learning in Artificial Neural Networks The learning process of an artificial neural network involves adjusting the weights and biases associated with each neuron to minimize the error between the network's predicted output and the actual target values. This process is typically conducted through supervised learning, where the network is trained using a labeled dataset. The main steps involved in the learning process are as follows:

- **Forward Propagation**: In forward propagation, the input data is passed through the network layer by layer to compute the output. This involves calculating the weighted sums and applying the activation functions at each neuron.

- **Loss Function**: The loss function quantifies the difference between the predicted output of the network and the actual target values. Common loss functions include mean squared error (MSE) for regression tasks and cross-entropy loss for classification tasks. Mathematically, for a single data point, the loss function L can be defined as:

$$L = (y_{target} - y_{pred})^2 \tag{9}$$

where y_{target} is the actual target value and y_{pred} is the predicted output.

Mean Squared Error (MSE): For regression tasks, the MSE loss function is defined as

$$L_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_{target,i} - y_{pred,i})^2 \tag{10}$$

where $y_{target,i}$ is the actual target value, $y_{pred,i}$ is the predicted output, and N is the number of data points.

Cross-Entropy Loss: For classification tasks, the cross-entropy loss function is defined as

$$L_{CE} = -\sum_{i=1}^{N} y_{target,i}\log(y_{pred,i}) \tag{11}$$

where $y_{target,i}$ is the true label (usually 0 or 1), and $y_{pred,i}$ is the predicted probability of the positive class.

- **Backpropagation**: Backpropagation [22] is the process of computing the gradient of the loss function with respect to each weight in the network, using the chain rule of calculus. This allows the network to determine how each weight contributes to the overall error.

- **Gradient Descent**: Gradient descent is an optimization algorithm used to update the weights and biases in the direction that minimizes the loss function. The weights are updated as follows:

$$w_{new} = w_{old} - \eta\frac{\partial L}{\partial w} = w_{old} + \eta\Delta w \tag{12}$$

where η is the learning rate, which controls the size of the weight updates, and $\frac{\partial L}{\partial w}$ represents the gradient of the loss function with respect to the weight w; and where Δ represents the computed gradient.

The learning rate η plays a crucial role in determining how quickly or slowly the network learns. A high learning rate may cause the network to converge too quickly to a suboptimal solution, while a low learning rate may result in slow convergence. Techniques such as adaptive learning rates and momentum are often used to improve the convergence of the training process.

Through this iterative process of forward propagation, loss calculation, backpropagation, and weight update, the neural network gradually learns to approximate the desired mapping between inputs and outputs, enabling it to make accurate predictions on new, unseen data.

2.1.2 Deep Learning

Non-deep artificial neural networks, also known as shallow neural networks, present several limitations when compared to deeper architectures. While shallow neural networks possess the capacity to approximate linear and simple non-linear relationships effectively, they struggle to capture intricate, high-dimensional patterns due to their limited depth. In contrast, deep neural networks, with their multiple hidden layers, are better equipped to learn hierarchical features and represent complex functions in a more efficient manner. Shallow networks often require an excessive number of neurons in a single hidden layer to approximate complex functions adequately, which can result in increased computational complexity and memory demands. Deep networks, however, mitigate this issue by distributing the learning process across multiple layers, thereby enhancing scalability and efficiency for handling complex tasks.

Another significant limitation of shallow networks is their proneness to overfitting, particularly when trained with a limited dataset. Overfitting occurs when the network learns to memorize the training data instead of generalizing to new, unseen data. Shallow architectures, with their restricted depth and flexibility, are more susceptible to overfitting. Conversely, deep networks are better suited to generalization due to their capacity to learn more abstract representations. Moreover, regularization techniques such as dropout prove more effective in deeper architectures.

These drawbacks underscore the importance of deep architectures in addressing complex tasks. Deeper networks are generally more suitable for learning rich, hierarchical representations that are essential for various real-world applications, such as image recognition [8], natural language processing [27], and speech recognition [9].

Deep neural networks (DNNs) are a type of artificial neural network characterized by multiple hidden layers between the input and output layers. The additional hidden layers enable the network to learn increasingly abstract and complex features from the input data, thereby allowing DNNs to solve more complex tasks compared to shallow networks. Each layer in a DNN captures a different level of representation, from low-level features such as edges in an image to high-level concepts such as objects. This hierarchical learning capability makes deep networks highly effective for tasks like image classification, language modeling, and speech recognition.

The increased depth of a DNN, however, introduces challenges, such as overfitting and increased computational cost. Various techniques have been developed to address these challenges, including weight regularization, early stopping, and the use of training, validation, and test datasets.

Weight Decay Weight decay, also known as L_2 regularization, is a technique used to prevent overfitting in neural networks by penalizing large weights. The regularization term is added to the loss function, which is proportional to the sum of the squared weights:

$$L' = L + \lambda \sum_{i=1}^{n} w_i^2 \,, \tag{13}$$

where L is the original loss function, w_i represents the weights of the network, and λ is a regularization parameter that controls the strength of the penalty. By discour-

aging the model from relying excessively on individual neurons with large weights, weight decay helps improve the model's generalization ability, thereby reducing the risk of overfitting.

Early Stopping Early stopping is another regularization technique used to mitigate overfitting by terminating the training process when the model's performance on a validation set begins to deteriorate. During training, the model's performance is monitored on both the training set and a separate validation set. When the validation loss ceases to decrease and starts increasing, it indicates that the model is beginning to overfit the training data. At this point, training is stopped to ensure that the model retains good generalization to new data.

Training, Validation, and Test Datasets To properly evaluate the performance of a neural network and ensure that it generalizes well to new, unseen data, the available dataset is typically divided into three subsets: the training set, the validation set, and the test set.

- The **training set** is used to train the model, meaning that the weights and biases are adjusted based on this data to minimize the loss function.
- The **validation set** is used to tune hyperparameters, such as the learning rate, regularization parameter, or network architecture. It is also used for early stopping, as it provides an unbiased evaluation of the model during training.
- The **test set** is used to evaluate the final performance of the model after training is complete. It provides an indication of how well the model is expected to perform on new, unseen data.

By splitting the dataset into these three subsets, it is possible to ensure that the model neither overfits to the training data nor underperforms on new data, thereby enabling a fair assessment of its generalization capabilities.

Despite their success, classical DNNs exhibit limitations when dealing with structured data. Structured data, which is organized in a defined schema, often possesses inherent spatial or temporal relationships — such as the arrangement of pixels in an image or the sequence in time-series data — that classical DNNs are not well-equipped to handle. Furthermore, redundancy within structured data, like the similar values in neighboring pixels of an image, can hinder classical DNNs from effectively recognizing patterns. To address these limitations, specialized architectures such as Convolutional Neural Networks [20] (CNNs) and Recurrent Neural Networks [16] (RNNs) have been developed, which explicitly account for spatial and temporal dependencies, respectively.

2.1.3 Convolutional Neural Networks and Recurrent Neural Networks

Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are two widely used architectures in the field of deep learning, each with distinct capabilities and purposes. CNNs are primarily utilized for image and spatial data, excelling in tasks that involve visual recognition and processing. RNNs, on the

other hand, are designed to handle sequential data, making them ideal for time-series predictions, natural language processing, and other tasks involving temporal dependencies.

Mathematical Convolution Convolution is a fundamental operation used in many fields, including image processing and signal analysis, and is the key operation in Convolutional Neural Networks. Mathematically, convolution is defined as follows:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau\,, \tag{14}$$

where $f(\tau)$ and $g(t-\tau)$ are functions defined over continuous variables, and $(f * g)(t)$ represents the result of the convolution. In the discrete domain, which is used in CNNs, the convolution is defined as:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n-m]\,, \tag{15}$$

where f and g are discrete signals, and the resulting $(f * g)[n]$ is the output of the convolution operation.

Convolution involves combining two functions or signals to produce a third function, which represents the overlap between the two signals as they shift over each other. In the context of CNNs, this operation is used to extract features from input data by applying filters *(kernels)* to the input, creating feature maps that represent different aspects of the input data.

Convolutional Neural Network (CNN) Convolutional neural networks are distinguished from other types of neural networks by their exceptional performance with structured data, such as images, audio, or other spatially correlated signals. CNNs consist of three main types of layers:

- **Convolutional layer**
- **Pooling layer**
- **Fully-connected (FC) layer**

The **convolutional layer** is the first layer of a convolutional network and is responsible for performing the convolution operation, which involves the use of filters to extract meaningful features from the input. This layer can be followed by additional convolutional or pooling layers to refine and distill the features further. The final layer is typically a fully-connected layer, which aggregates the learned features to produce the output.

In a convolutional layer, the essential components are the input data, the filter *(or kernel)*, and the feature map *(or activation map)*. The filter is a small matrix of weights (often of size 3×3 or 5×5), which is applied to a receptive field of the input data to compute a dot product, yielding a single value in the output feature

map. This operation is repeated as the filter shifts (or *strides*) across the input, producing the entire feature map:

$$\text{Feature Map Value} = \sum_{i=1}^{M} \sum_{j=1}^{N} I_{i,j} \cdot K_{i,j} \,, \tag{16}$$

where $I_{i,j}$ represents the pixel value from the input image, and $K_{i,j}$ represents the filter weights. The stride controls the step size of the filter, and padding may be added to preserve the spatial dimensions of the input.

One key concept in CNNs is **parameter sharing**, where the weights of the filter remain fixed as it is applied across the entire input. This not only reduces the number of parameters but also allows the network to learn spatial hierarchies effectively. The weights in the filters are adjusted through training using *backpropagation* and *gradient descent*.

The **pooling layer** is used to reduce the spatial dimensions of the feature maps, thereby reducing the number of parameters and computational complexity. Common pooling operations include max pooling and average pooling.

Limitations of Convolutional Neural Networks While CNNs are powerful tools for image and spatial data processing, they have several limitations:

- **Large Data Requirements**: CNNs require a substantial amount of labeled data to achieve high performance. Training a CNN on small datasets can lead to overfitting and poor generalization.
- **High Computational Cost**: The training process of CNNs is computationally expensive, especially for deep architectures with many layers. This requires powerful hardware, such as GPUs, and significant time for training.
- **Limited Ability to Handle Temporal Dependencies**: CNNs are not inherently designed to capture temporal dependencies or sequential data, which limits their application in tasks like language modeling or time-series prediction.
- **Sensitivity to Adversarial Attacks**: CNNs are vulnerable to adversarial attacks, where small, imperceptible changes to the input can lead to incorrect predictions.

Recurrent Neural Network (RNN) Recurrent neural networks are designed to model sequential data by maintaining a form of memory, which allows information from previous time steps to influence the current output. Unlike feedforward networks, RNNs have recurrent connections, which enable them to retain context over a sequence of inputs.

The distinguishing feature of RNNs is that they share parameters across each time step. While traditional deep learning networks assume that inputs and outputs are independent of each other, RNNs explicitly consider the dependency between

successive elements of a sequence. This characteristic allows RNNs to excel in applications such as language modeling, where each word in a sentence depends on the preceding words.

Mathematically, the hidden state h_t at time step t is computed as follows:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)\,, \tag{17}$$

where W_{hh} represents the weight matrix for the hidden state, W_{xh} is the weight matrix for the input, x_t is the input at time t, b_h is the bias term, and tanh is an activation function that introduces non-linearity.

RNNs are trained using **backpropagation through time (BPTT)**, which is a variant of the traditional backpropagation algorithm adapted for sequences. In BPTT, the error is propagated backward through each time step, and the gradients are accumulated to adjust the parameters appropriately. This enables the network to learn long-range dependencies within the sequence.

However, standard RNNs face challenges in learning long-term dependencies due to the problem of vanishing or exploding gradients. To mitigate these issues, advanced variants such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) are employed, which include mechanisms to control the flow of information through the network, thereby allowing them to maintain relevant information over extended sequences.

Limitations of Recurrent Neural Networks RNNs, despite their effectiveness in handling sequential data, have certain limitations:

- **Vanishing and Exploding Gradients**: Standard RNNs struggle with learning long-term dependencies due to the vanishing or exploding gradient problem, which makes training difficult for long sequences.
- **High Computational Cost**: Training RNNs, especially with long sequences, is computationally intensive due to the need for backpropagation through time, which requires unrolling the network for each time step.
- **Difficulty in Parallelization**: Unlike feedforward networks, RNNs process data sequentially, making it challenging to parallelize computations, which can lead to longer training times.
- **Short-Term Memory**: Standard RNNs tend to have difficulty retaining information over long sequences, which limits their ability to capture long-term dependencies. LSTMs and GRUs were introduced to address this limitation, but they come with increased model complexity.

In summary, CNNs and RNNs are powerful architectures for processing spatial and temporal data, respectively. CNNs excel in feature extraction from spatial data, while RNNs are well-suited for modeling dependencies in sequential data, each providing foundational capabilities for various applications in deep learning.

2.1.4 Transformer

The transformer model[25] represents a major breakthrough in neural network architecture, eliminating the need for traditional approaches like recurrent neural networks (RNNs) or convolutional neural networks (CNNs), which come with significant limitations. RNNs, for instance, rely on sequential processing, making them slow and computationally expensive, especially with long sequences. CNNs, on the other hand, are typically more suited to image processing and struggle to capture long-term dependencies in sequences. Transformers, however, take a completely different approach: they process entire input sequences in parallel, allowing for far greater efficiency in both training and inference stages. Unlike previous architectures, simply adding more GPUs does not linearly accelerate RNNs, while transformers benefit much more from parallelization, enabling them to train faster and more efficiently.

At its core, a transformer is a neural network that learns context and meaning by analyzing relationships within sequential data, such as words in a sentence or data points in time-series data. The central innovation of transformers is the **self-attention mechanism**, a powerful mathematical framework that allows the model to weigh the relevance of each element in a sequence in relation to every other element. This self-attention capability enables transformers to detect and model complex dependencies and patterns, even across long distances within the sequence. For example, in a sentence, a transformer can easily understand the relationship between a noun at the beginning and an adjective at the end, regardless of the words in between.

The transformer model architecture is built on a series of layers, each containing a self-attention mechanism and a feedforward neural network. The self-attention mechanism scans the input data, which can be sequences of tokens (like words or subwords) or other structured data, and assigns varying degrees of importance to each element based on its relationship to other elements. This allows the model to dynamically adjust its focus and attention based on context. The feedforward network processes the output of the self-attention mechanism, enabling the model to transform and refine its internal representations of the data. Multiple layers are stacked together, allowing the model to progressively build a rich, nuanced understanding of the input.

Mathematically, the **self-attention** mechanism computes attention scores as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V\,, \tag{18}$$

where Q, K, and V represent the query, key, and value matrices, respectively, and d_k is the dimensionality of the key vectors. The softmax function is used to normalize the attention scores, ensuring they sum to 1. This allows the model to focus on different parts of the input sequence with varying importance.

Because transformers can capture both short- and long-range dependencies, they have quickly become the foundation of numerous advanced applications, especially in natural language processing, like machine translation, text generation, and sentiment analysis. Their ability to parallelize computations makes them highly scalable, enabling training on massive datasets and achieving groundbreaking performance. This architecture has not only set a new standard for efficiency but has also paved

the way for developing large-scale language models, revolutionizing how we approach tasks in machine learning and artificial intelligence.

As far as the foundational models we used go, Segment Anything, DinoV2, and Open Clip are based on transformers.

2.1.5 Zero-shot learning

Zero-shot learning[23] is an advanced machine learning technique where a model can identify and classify new concepts without any labeled examples, which is why it is termed "zero-shot." This approach relies on transferring knowledge gained from pre-training on extensive, diverse, unlabeled datasets. During pre-training, models are exposed to varied data, such as images, text, or both, allowing them to develop a comprehensive understanding of the world through rich visual and semantic representations. These representations capture relationships between objects, their attributes, and associated contextual information, forming a versatile knowledge base that models draw upon during zero-shot tasks. In zero-shot learning, the model leverages its pre-trained knowledge to make educated inferences. When presented with a new class, it examines the class description or attributes and maps them onto its learned feature space, allowing it to recognize whether an unseen instance aligns with this concept. The model can accomplish this without directly encountering labeled samples of the new class.

By removing the dependency on labeled training data, zero-shot learning enables models to generalize to new concepts on the fly, purely through descriptive cues. This flexibility is invaluable for real-world AI applications, where the need to adapt quickly to new data types, topics, or categories is paramount. The practical implications of zero-shot learning are vast. It allows models to extend to novel tasks, products, and markets without the time and expense of gathering and labeling data or retraining. As new needs or trends emerge, a zero-shot model can dynamically recognize and categorize an open-ended range of new concepts using only high-level descriptions or specifications. This adaptability helps organizations drive cost-effective innovation, personalize their offerings, assess risks, and detect anomalies in ways that are both scalable and flexible. Zero-shot learning is, therefore, instrumental in building AI that can evolve in alignment with fast-changing business environments, creating future-proof AI solutions that stay relevant and effective over time. By enabling scalable classification without the need for labeled examples, zero-shot learning is advancing the field toward more generalized machine learning. This adaptability not only reduces operational costs but also paves the way for AI systems that can keep up with the rapid pace of modern industries, making zero-shot learning a key component of next-generation, dynamic artificial intelligence.

2.2 Foundational Models

With a foundational understanding of machine learning established, it is now appropriate to introduce the core models utilized in this study to achieve our overarching objective. Specifically, the aim is to enable zero-shot querying within a reconstructed 3D mesh of an environment and retrieve coherent and meaningful results. This process involves leveraging advanced machine learning architectures to bridge the gap between high-dimensional spatial data and the semantic reasoning required for effective querying. The models and methodologies detailed in the subsequent sections form the backbone of this approach, addressing both the technical challenges and the conceptual innovations necessary for achieving this goal.

The models employed to reach this thesis goal are: SAM [14] (Segment Anything Model), DINOV2 (DIstillation with NO Labels) [1, 19] and open CLIP [21, 18] (Contrastive Language–Image Pre-training). These models are used to perform the different steps to go from the raw data to a reconstruction of the 3-dimensional environment to querying such environment.

CLIP Contrastive Language-Image Pretraining is used for associating visual data with natural language. CLIP's ability to learn a multi-modal embedding space from image-text pairs enables the understanding and interpretation of visual scenes through textual descriptions. This capability is vital for tasks that require the system to recognize and categorize objects or scenes based on natural language prompts, facilitating better human-machine interaction and zero-shot learning.

DINO Distillation with No Labels is a self-supervised learning of visual representations. DINO's role is to generate robust and generalizable visual features without the need for labeled data. These features are then used in various downstream tasks such as object detection, classification, and segmentation. DINO helps in building a strong visual understanding of the environment, which is essential for tasks that involve recognizing and interpreting visual information in diverse and unseen scenarios.

SAM Segment Anything Model is used for segmentation tasks. It is designed to produce accurate segmentation masks in response to various prompts, whether they involve specific objects or regions within an image. SAM's ability to handle ambiguous prompts and generate multiple valid masks with confidence scores makes it invaluable for tasks that require precise identification and separation of different elements in a scene. SAM is crucial for segmenting the environment into meaningful parts, enabling better object recognition and interaction.

Together, these models enable to achieve a comprehensive understanding of both the spatial layout and the semantic content of its environment, facilitating a wide range of complex tasks in robotics, autonomous systems, and beyond.

2.2.1 SAM

In Natural Language Processing (NLP) and computer vision, foundation models mark a significant leap forward, enabling capabilities like zero-shot and few-shot

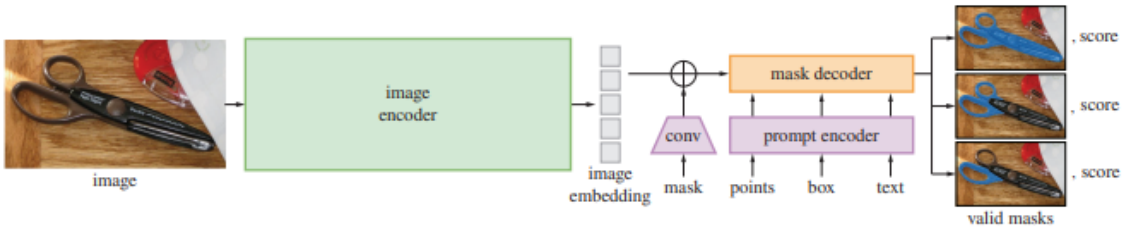


Figure 3: Segment Anything Model (SAM) overview: A powerful image encoder generates an image embedding that can be efficiently queried by various input prompts to produce object masks at near real-time speeds. When faced with ambiguous prompts that correspond to multiple objects, SAM can generate multiple valid masks along with corresponding confidence scores. Image from [14]

learning for new datasets and tasks, often utilizing "prompting" techniques. Building on this concept, the Segment Anything paper [14] presents the idea of promptable segmentation, which aims to generate precise segmentation masks in response to any given prompt. A prompt specifies the target for segmentation within an image, possibly including spatial or textual hints to identify an object. The requirement for a valid mask ensures that the model generates an output even when the prompt is ambiguous and could refer to multiple objects.

The promptable segmentation task, coupled with the goal of practical application, imposes particular demands on the model's architecture. The model must be versatile enough to handle different prompts, able to generate masks in real-time for interactive usage, and capable of dealing with ambiguities in the input. Notably, we found that a simple architecture meets all these criteria: a powerful image encoder produces image embeddings, a prompt encoder interprets the prompts, and these are combined in a lightweight mask decoder to generate segmentation masks. This architecture is known as the Segment Anything Model (SAM).

To achieve strong generalization to new and varied data distributions, SAM is trained on a vast and diverse array of masks, going beyond what is offered by existing segmentation datasets. Unlike other foundation models that rely on abundant online data, the scarcity of masks necessitates a different strategy. The solution presented in the paper is a "data engine," a system co-developed with the model that includes model-in-the-loop dataset annotation. This data engine progresses through three stages: assisted-manual, semi-automatic, and fully automatic. In the initial stage, SAM assists human annotators, similar to traditional interactive segmentation. In the semi-automatic stage, SAM generates masks for certain objects based on likely locations, allowing annotators to concentrate on less frequent objects, thereby increasing mask diversity. Finally, in the fully automatic stage, SAM is prompted with a grid of foreground points, yielding an average of about 100 high-quality masks per image.

SAM's design is largely driven by the need for efficiency. With image embeddings already computed, the prompt encoder and mask decoder can run within a web browser on a CPU, with a response time of around 50 milliseconds. This efficiency enables smooth, real-time interaction with the model.

During training, mask predictions are supervised using a combination of focal

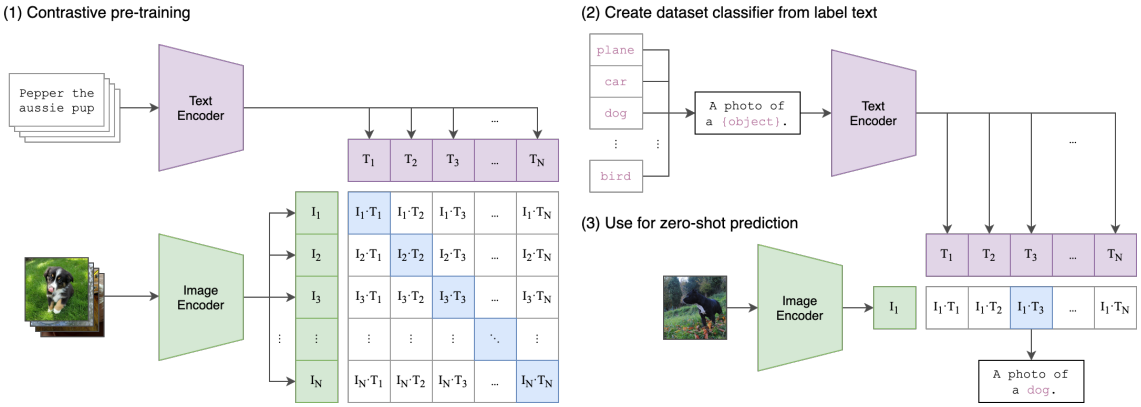


Figure 4: The CLIP approach as presented in the CLIP paper [21] and CLIP GitHub [18]

loss and dice loss. The promptable segmentation task is trained using a variety of geometric prompts, simulating an interactive scenario by randomly sampling prompts over multiple rounds per mask. This method allows SAM to seamlessly integrate into the data engine.

2.2.2 CLIP

CLIP (Contrastive Language–Image Pre-training) [21, 18] builds on extensive research in zero-shot transfer, natural language supervision, and multimodal learning. The concept of zero-data learning emerged over a decade ago, initially focused on computer vision as a method for recognizing new object categories. A key advancement was recognizing that natural language could serve as a versatile prediction framework, facilitating broader generalization and transfer capabilities. Given a batch of N (image, text) pairs, the CLIP [21, 18] model is trained to discern which of the $N \times N$ potential (image, text) combinations within the batch actually occurred. To achieve this, CLIP learns a multi-modal embedding space by simultaneously training an image encoder and a text encoder. The objective is to maximize the cosine similarity between the embeddings of the correct N pairs in the batch while minimizing the cosine similarity for the $N^2 - N$ incorrect pairings. The training process optimizes a symmetric cross-entropy loss over these similarity scores.

Thanks to the extensive size of the pre-training dataset, overfitting is not a significant concern, allowing for a streamlined training process. Unlike some previous approaches, CLIP is trained from scratch without the need to initialize the image encoder with pre-trained ImageNet weights or the text encoder with any pre-existing weights. Furthermore, the model does not employ a non-linear projection between the representation space and the contrastive embedding space. Instead, CLIP uses a straightforward linear projection to map the outputs from each encoder into the multi-modal embedding space. During training, this simpler approach showed no loss in efficiency compared to more complex methods. It is speculated that non-linear projections may be particularly suited to specific self-supervised learning methods that are tailored for images alone.

In addition to these simplifications, CLIP omits the text transformation function

used in prior models, which uniformly samples a single sentence from the text. This adjustment was made because many of the (image, text) pairs in CLIP's pre-training dataset are already composed of single sentences. The image transformation function is also simplified; the only data augmentation applied is a random square crop from resized images. Lastly, the temperature parameter, which influences the range in the softmax function, is optimized directly during training as a log-parameterized multiplicative scalar, removing the need to tune it as a hyper-parameter.

CLIP's pre-training is focused on determining whether an image and a text snippet are paired together in its dataset. This ability can be directly applied to zero-shot classification. For any given dataset, the model uses the names of the classes as the set of possible text pairings and predicts the most likely (image, text) pair according to its learned embeddings. Specifically, CLIP first computes the feature embedding of the image and the embeddings of the set of potential texts using their respective encoders. The cosine similarity between these embeddings is then calculated, scaled by the temperature parameter, and normalized into a probability distribution via softmax.

This prediction layer operates as a multinomial logistic regression classifier with normalized inputs, normalized weights, no bias, and temperature scaling. In this framework, the image encoder serves as the backbone for computer vision tasks, generating a feature representation of the image. The text encoder, functioning as a hypernetwork, generates the weights of a linear classifier based on the text, which specifies the visual concepts that the classes represent.

Each step of CLIP's pre-training can be viewed as optimizing the performance of a randomly constructed proxy for a computer vision dataset, one that hypothetically contains one example per class and a total of 32,768 classes defined through natural language descriptions. For zero-shot evaluation, once the zero-shot classifier is generated by the text encoder, it is cached and reused for all subsequent predictions, which spreads the computational cost across all predictions in a dataset.

CLIP models learn to perform a wide range of tasks during pre-training, and this capability can be harnessed via natural language prompting to enable zero-shot transfer to many existing datasets. When scaled sufficiently, this approach can deliver performance that rivals task-specific supervised models, though there remains substantial room for improvement in this area.

2.2.3 DINO

Given a batch of paired images, DINO [1] (Distillation with No Labels) is trained to produce meaningful visual representations without the need for explicit supervision. DINO operates by distilling knowledge from a teacher network to a student network, both of which share the same architecture but have different parameters. The goal is to train the student network to match the teacher's output, effectively transferring knowledge from one to the other. This process involves learning representations by maximizing the similarity between the teacher and student outputs for the same image while minimizing it for different images. The resulting representations are optimized using a cross-entropy loss function that measures the discrepancy between the teacher and student predictions.

One of the key aspects of DINO's training process is the absence of labeled data,

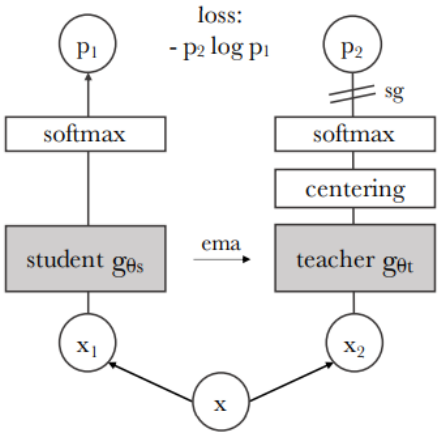


Figure 5: DINO can be illustrated using a single pair of image views $(x1, x2)$. Two different random transformations of the same image are passed through the student and teacher networks, which share the same architecture but have different parameters. The teacher network's output is centered using the batch mean, and both networks produce a K-dimensional feature vector normalized with a temperature softmax. The similarity between the outputs is measured using cross-entropy loss. Gradients are only propagated through the student network using a stop-gradient (sg) operator on the teacher. The teacher's parameters are updated using an exponential moving average (ema) of the student's parameters.

allowing the model to learn in a self-supervised manner. Unlike traditional models that require large amounts of annotated data, DINO leverages the intrinsic structure of the data itself to learn useful features. The teacher network's parameters are updated using an exponential moving average of the student network's parameters, ensuring that the teacher's knowledge evolves smoothly over time. This setup allows the student network to continually improve by learning from the teacher's increasingly refined representations.

DINO's training involves a set of image augmentations, where multiple views of the same image are generated through random crops, color jittering, and other transformations. Both the teacher and student networks process these augmented views, and the model is trained to ensure that corresponding views produce similar outputs. By using diverse augmentations, DINO encourages the networks to learn representations that are invariant to these changes, resulting in more robust visual features.

A notable feature of DINO is its ability to produce high-quality attention maps, which highlight the most relevant parts of an image for a given task. These attention maps are generated by the student network and can be visualized to understand which areas of the image the model is focusing on. This capability is particularly useful in tasks like object detection and segmentation, where understanding the spatial focus of the model is crucial. For zero-shot transfer, DINO's learned features are used directly for downstream tasks such as image classification, object detection, and semantic segmentation. The model's ability to learn without labeled data means it can be applied to new datasets with minimal adjustments, making it highly adaptable.

Sensor type	Hardware component
Head tracking	4 visible light camera
Eye tracking	2 IR cameras
Depth	1-MP time-of-flight (ToF) depth sensor
MU	Accelerometer, gyroscope, magnetometer
Camera	8-MP stills, 1080p30 video
Audio	Microphone array 5 channels

Table 1: HoloLens 2 sensors

2.3 HoloLens 2

The HoloLens 2 is a mixed reality (MR) headset developed by Microsoft that integrates augmented reality (AR) with real-world environments. It is primarily used for various professional, educational, and industrial applications due to its ability to overlay digital information onto the physical world, enhancing the user's interaction with digital content. HoloLens 2 can display such information due to it being equipped with advanced sensors, including depth sensors, inertial sensors, and eye-tracking technology, enabling precise tracking of the user's movements and gaze.

In this thesis project, the use of the HoloLens system has been chosen due to two main advantages, the system provides: the high-quality data stream it can produce and the possibility to use the visor to reconstruct real-world environment; which are both fundamental qualities for our task.

To use the data produced by HoloLens 2, the Research Mode [24] had to be enabled, which granted access to the raw streams on the device. Using such sensors, combined with the Research Mode, we were able to scan the real world and map it into a point cloud in an easy and fast manner.

The resulting data from the use of HoloLens 2, in Research Mode to perform a scan, are then processed using the StreamRecorderConverter from the Microsoft HoloLens repository [24]. The data produced are:

- `PV/`: which is a folder that contains the images in a raw format `*.byte` and in *.png
- `Depth Long Throw/`: which is a folder that contains the generated point clouds in `*.ply`, `*.pmg`
- `pinhole_projection/`: which is a folder that contains
  - `rgb/`: which is a folder containing the rgb images `*.png`
  - `depth/`: which is a folder containing the depths images `*.png`
  - `calibration.txt`: which contains the calibration parameters of the camera.
  - `depth.txt`: which contains the association between Depth Long Throw timestamp and depth image `*.png` timestamp.

  - `rgb.txt`: which contains the association between Depth Long Throw timestamp and rgb image `*.png` timestamp
  - `odometry.log`: which is composed by 4x4 matrix of float representing the camera pose
  - `trajectory.xyz`: which contains a set of 3d coordinates

- `Depth Long Throw_rig2world.txt`: which contains, for each image, as in the following order
  - timestamp
  - the other data, on the row, compose a 4x4 matrix of extrinsics

- `*_pv.txt`: which contains data as in the following order
  - the x-coordinate of the principal point
  - the y-coordinate of the principal point
  - image width
  - image height

  And for each image
  - timestamp
  - focal length along the x-axis
  - focal length along the y-axis
  - the other data, on the row, compose a 4x4 matrix of extrinsics

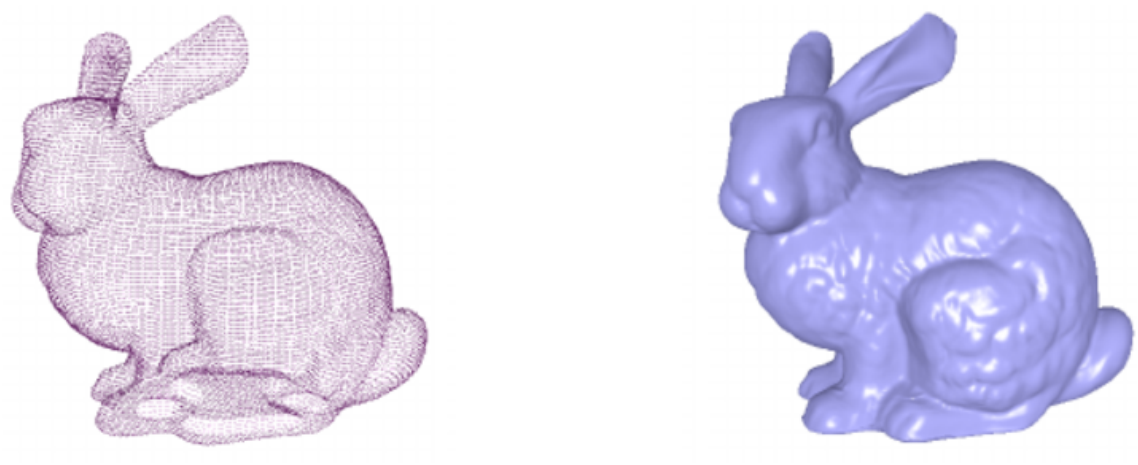
- `Depth Long Throw_extrinsics.txt`: which contains the values to compose a 4x4 matrix of extrinsics
- `Depth Long Throw_lut.bin`: which contains a series of 3d coordinates.
- `2023-12-12-105500_head_hand_eye.csv`: which contains hand tracking and eye gaze tracking results projected on PV images.

2.4 Photogrammetry and Point-clouds

The data captured using the HoloLens 2 is leveraged to conduct photogrammetry, ultimately generating a point cloud. Photogrammetry[4][6] is a technique in which multiple overlapping images are captured from different angles and perspectives to create a digital, three-dimensional representation of real-world objects or environments. By carefully photographing a subject from various vantage points, photogrammetry software can analyze the images to reconstruct detailed 3D models, enabling in-depth examination, measurement, and analysis. Unlike traditional 3D scanning methods that rely on laser technology or structured light to map a scene, photogrammetry uses real photographs to capture the textures, colors, and shapes of objects, rendering them into 3D models. This reliance on photographs means

that the quality of the resulting 3D model is directly influenced by the quality and coverage of the image dataset. High-resolution images, optimal lighting conditions, and systematic coverage are essential for producing an accurate and detailed model. Photogrammetry requires a structured approach, ensuring that images are taken systematically to capture all relevant aspects of the subject, whether it's an archaeological site, architectural structure, historical artifact, or industrial object.

The final output of photogrammetry is often a point cloud, a collection of data points in three-dimensional space that represent the external surface of the scanned object or environment. Each point corresponds to a specific position on the object's surface, creating a precise spatial map. Point clouds serve as a foundational dataset that can be used for further processing, including mesh generation, surface reconstruction, and texture mapping. This data is invaluable in fields like cultural heritage preservation, virtual reality, engineering, and environmental monitoring, where precise spatial information about real-world subjects is needed for accurate digital replication and analysis. By utilizing photogrammetry and point-cloud generation, detailed 3D models of real-world subjects can be created without invasive or contact-based scanning methods, making it a preferred approach for delicate or historically significant objects. The HoloLens 2 aids this process by providing spatial mapping capabilities, enhancing the accuracy of the photogrammetry workflow, and enabling the creation of high-quality, interactive 3D models directly from the field.



(a) Bunny Point-cloud (b) Bunny Mesh

Figure 6: An example of a Point-cloud and a Mesh

2.5 Ray Tracing

Ray tracing[7] is a highly sophisticated rendering technique used to simulate lighting in a scene with remarkable realism, capturing complex phenomena like reflections, refractions, shadows, and indirect lighting. This method works by tracing the path of light rays as they travel from a virtual camera through a 2D viewing plane (or pixel plane) and into a 3D scene, ultimately reaching back to the various light sources. By accurately modeling the way light interacts with surfaces, ray tracing produces

lifelike visuals that closely mirror the way our eyes perceive the real world. This technique is widely used in computer graphics, serving both non-real-time applications, such as film and television production, where high-quality, photorealistic rendering is essential, and real-time applications like video games, where recent advancements in hardware have made real-time ray tracing feasible. Beyond entertainment, ray tracing finds applications in fields like architecture, engineering, and lighting design, where accurate light behavior simulation is essential for visualization and analysis.

A core component of ray tracing is ray casting, a process in which rays are emitted from the camera (or eye) position through each pixel in the image plane to determine intersections with objects in the scene, typically represented by primitives like triangles. When a ray intersects a primitive, the algorithm calculates the distance from the camera to the object and retrieves the object's color and material data, which then contributes to the final color of the pixel. This process is repeated for each pixel, combining data from intersected surfaces, light sources, and additional rays to simulate complex lighting effects, resulting in a highly realistic image.

2.6 VCGlib and Intel Embree

The Visualization and Computer Graphics Library (VCGlib)[3] is an open-source, portable C++ templated library designed for the manipulation, processing, and visualization of triangle and tetrahedral meshes, with OpenGL support for displaying these structures. Primarily focused on triangular meshes, VCGlib provides an extensive set of features for advanced mesh processing, including operations for mesh simplification, optimization, and manipulation. It serves as the core computational foundation for MeshLab [2], a popular open-source tool for 3D mesh processing and visualization.

Within the VCGlib framework, I developed an adapter, EmbreeAdaptor, which enables seamless integration with Intel Embree [26], a high-performance ray tracing library created by Intel. Embree is optimized for speed and accuracy in ray tracing tasks and is widely used in professional applications that demand fast, high-quality rendering. The EmbreeAdaptor allows users to leverage Embree's efficient ray tracing capabilities within VCGlib, facilitating high-performance intersection testing and scene rendering. By combining VCGlib's mesh processing capabilities with Embree's ray tracing algorithms, the adapter creates a streamlined, powerful tool for applications requiring precise and efficient ray-object intersection computation.

Chapter 3 Related Work

After defining the objective of this thesis project — to reconstruct a 3D environment that can be queried in a zero-shot manner — we conducted a comprehensive literature review to identify existing models and techniques that align with or support our approach. This involved exploring various methods of 3D reconstruction and querying, with a specific focus on models that enable zero-shot capabilities. Our review aimed to uncover how current advancements in 3D modeling, scene understanding, and knowledge transfer could be adapted or extended to meet the unique requirements of our project.

We examined state-of-the-art visual computing techniques, focusing on models that integrate semantic understanding with spatial information to facilitate effective querying over 3D reconstructions. Through this review, we identified existing gaps and potential strategies for achieving a model capable of both detailed 3D environment reconstruction and zero-shot querying, forming the basis for the design and development of our system.

One of the papers we found, exploring a similar research branch, is the "ConceptFusion: Open-set Multimodal 3D Mapping" [11], which is an advanced approach to creating detailed 3D maps of environments by combining multiple types of data (or "modalities"), like vision, audio, and textual information, in a way that enables open-set recognition. Open-set recognition means that the model can handle and adapt to unfamiliar objects or categories it hasn't been explicitly trained on.

Another paper, with a similar aim to ours, is "3D Concept Learning and Reasoning from Multi-View Images" [10], which shares some similarities with ConceptFusion, especially in terms of creating a 3D understanding of the world from visual data. However, it focuses more specifically on building 3D representations of objects and reasoning about their properties by using images from multiple viewpoints.

A more recent approach, again with a similar purpose to our research, is the "LERF: Language Embedded Radiance Fields" [13]. LERF (Language Embedded Radiance Fields) represents a method that uses radiance fields embedded with language models to interactively generate and understand 3D scenes based on text inputs. This technique allows for the generation, modification, and comprehension of 3D environments by combining visual information with language-based embeddings, enabling a more interactive and intuitive approach to 3D scene manipulation.

3.1 The "ConceptFusion: Open-set Multimodal 3D Mapping" approach

The approach that the ConceptFusion paper presents is the following:

1. **Multimodal Data Fusion:** ConceptFusion combines data from multiple sensors, such as cameras, LiDAR, and depth sensors, to create a comprehensive 3D map of the environment. This multimodal approach enables the system to utilize different types of data (RGB images, depth information, etc.) to provide a more robust and complete understanding of the scene.

2. **Open-Set Recognition:** Unlike traditional recognition systems that are limited to identifying objects from a pre-defined set of categories, ConceptFusion is capable of open-set recognition. This means it can recognize objects and concepts that were not part of its initial training data. The system can continuously learn and adapt to new, previously unseen objects and environments, making it highly flexible and scalable.

3. **3D Semantic Mapping:** ConceptFusion constructs detailed 3D maps that are not just geometrically accurate but also enriched with semantic information. This means that the system can label objects and regions within the map with meaningful descriptors, such as identifying furniture, walls, doors, etc., in an indoor environment. The semantic information allows for a deeper understanding of the spatial layout and context of the scene.

4. **Real-Time Performance:** One of the key features of ConceptFusion is its ability to perform real-time mapping and understanding. This is crucial for AR applications, where timely and accurate information is necessary for interactive and immersive experiences. The system is designed to process incoming data streams quickly and update the 3D map dynamically.

Even though the approach presented in the ConceptFusion paper is very interesting and demonstrates promising results, some significant issues have been identified when we tried to apply the method to real-world data. The use of real-world data in ConceptFusion presents several challenges:

- **Lack of Dynamic Intrinsics Loading:** When using real-world data, the ConceptFusion paper does not provide a method for dynamically loading the camera intrinsics for each image. This limitation forces the use of a non-image-specific, average intrinsic matrix, which can lead to inaccuracies. In real-world scenarios, camera intrinsics often vary between frames or datasets due to different focal lengths, sensor sizes, or lens distortions. The lack of support for dynamically loading these parameters can reduce the fidelity of the results, especially when transitioning from synthetic datasets, where a single set of intrinsics might suffice, to more varied real-world datasets.

- **Inconsistent Reconstruction with Real-World Data:** When real-world RGB and depth images are captured to have the same or similar sizes as those used in the synthetic dataset, the ConceptFusion model is capable of generating results. However, these results are often inconsistent during the reconstruction phase. Additionally, when the model is queried, it fails to return high-quality answers and instead produces a noisy selection in the 3D environment. This inconsistency suggests that the model's performance is significantly affected by the quality and nature of the input data. The original paper's reliance on synthetic data may not generalize well to real-world data, where noise, varying lighting conditions, and sensor imperfections are prevalent.

- **High Memory Consumption with High-Resolution Data:** When using high-quality real-world data (e.g., images captured at 1920x1080 resolution), a

major issue encountered is the excessive amount of RAM required to compute an output. This issue arises primarily from the use of gradSLAM [15], which required around 183 optimization steps for convergence. During experimentation, it was observed that performing just 8 to 10 steps already saturated a system with 500GB of RAM. Due to this enormous amount of memory usage, we were unable to obtain any meaningful results using high-resolution data. This suggests that the current implementation of ConceptFusion is not optimized for handling high-resolution real-world datasets.

3.2 The "3D Concept Learning and Reasoning from Multi-View Images" approach

The "3D Concept Learning and Reasoning from Multi-View Images" [10] approach builds a 3D understanding of objects by learning from multiple 2D images taken from different angles. The typical process is composed of the following sequential steps:

1. **Collect Multi-View Images**
   The system captures 2D images of the target object from various angles, covering different perspectives and lighting conditions.
2. **Feature Extraction from Each View**
   Each image is processed using CNNs to extract key visual features, like edges and textures, with possible pre-processing to focus on the object.
3. **3D Representation Building (Fusion of Views)**
   Features from different views are fused to create a 3D model using voxel grids, point clouds, or implicit representations, often aided by neural networks.
4. **3D Concept Learning (Category-Level Understanding)**
   The model generalizes features to build a category-level concept, learning key characteristics that define the object class.
5. **Spatial Reasoning and Inference**
   With a learned concept, the model can infer spatial properties like orientation, parts, and functional relationships.
6. **Recognition and Categorization of New Instances**
   The trained model recognizes new object instances, generalizing its 3D knowledge to varied real-world appearances.

The approach of 3D Concept Learning and Reasoning from Multi-View Images is effective for building 3D models and understanding objects in three-dimensional space. However, it also faces several challenges and limitations:

- **Data Requirements and Coverage**
  Large amounts of 2D images from multiple angles are needed for accurate 3D reconstruction, which is time-consuming and sensitive to lighting and clarity issues.

- **Computational Complexity**
  Processing and fusing multi-view images into a 3D model is resource-intensive, limiting scalability, especially for real-time applications.

- **Generalization to New Categories**
  Recognizing objects from entirely new categories is difficult without further training, limiting flexibility in diverse applications.

These challenges indicate that while 3D Concept Learning and Reasoning from Multi-View Images is a promising approach, improvements in data acquisition, model efficiency, and handling of noise and occlusions are needed to enhance robustness and adaptability across diverse applications.

3.3 LERF: Language Embedded Radiance Fields

The LERF approach integrates language embeddings with neural radiance fields (NeRFs) to enable interactive 3D scene manipulation using natural language, comprising the following components:

1. **Language-Driven Scene Interaction:** LERF links text-based commands with 3D scene elements, allowing users to identify, modify, or query objects within a scene through natural language.

2. **Semantic Mapping of 3D Scenes:** Language embeddings align with visual features, enabling the model to understand the context within the 3D space and label objects or regions accordingly.

3. **Dynamic Scene Updates:** Supports real-time modifications based on textual input, enhancing usability in applications like virtual reality (VR) and augmented reality (AR).

While LERF presents a promising method for interactive 3D environments, challenges remain, such as generalizing across diverse and complex scenes and effectively linking varied language descriptions to specific spatial features.

Following the analysis of the current state-of-the-art systems, including the evaluation of their capabilities and limitations, we formulated the structure of our proposed system with the aim of addressing and mitigating the identified challenges. The rationale behind our approach, as well as the implementation details, are thoroughly explained in section 4. Additionally, the outcomes of our methodology, along with an in-depth discussion of the experimental results, are presented in section 5.

Chapter 4 Method

As stated before, the objective of this thesis project is to reconstruct a 3D environment that can be queried in a zero-shot manner. What this means is that we want to be able to query the environment and receive meaningful answers without the need for specific training of the model over the dataset we are querying.

To make such an objective possible, we propose and implement the following pipeline:

Phase 1, Environment Reconstruction: In this step, we perform a scan of the environment and elaborate the data to go from a set of point clouds to a 3D mesh: then we use the mesh and ray tracing to calculate the depth images.

Phase 2, Feature Extraction: Through this phase, we use a pretrained model capable of extracting the segmentation masks to feed a pretrained model capable of extracting the features: such features are then projected to the vertexes, one for each vertex, that form the mesh.

Phase 3, Querying: In the last phase, we establish a query and transform it to be comprehensible by our system; with a similarity metric, we check how close the query is to the different features assigned to the vertexes of the mesh. Using the similarity scores, we select and refine the results.

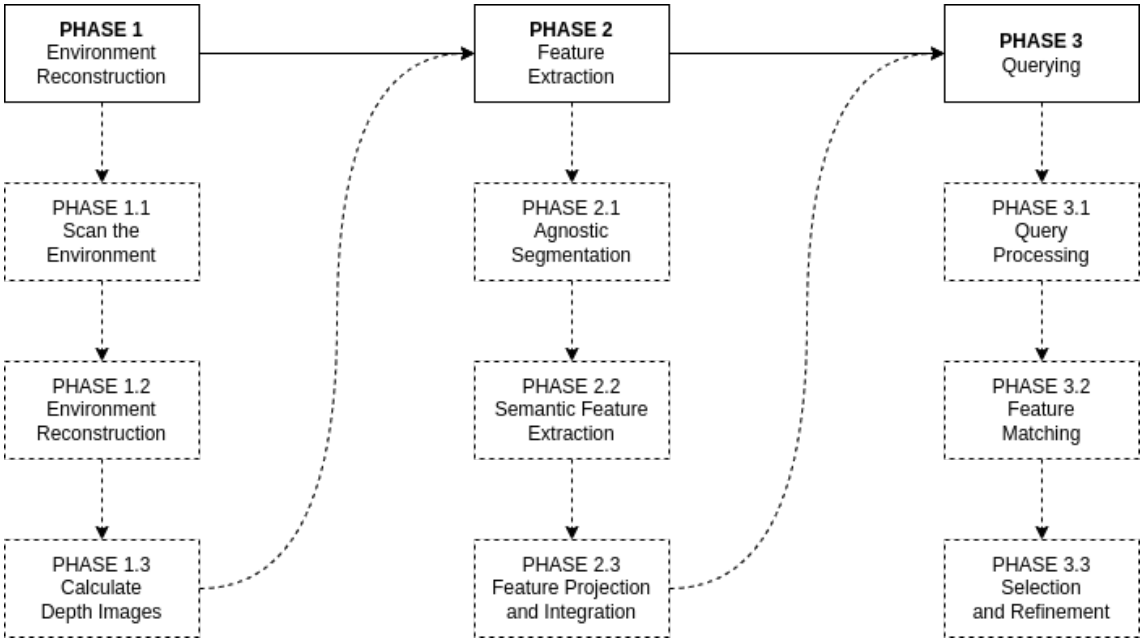


Figure 7: The workflow of our system

(a) Original RBG (b) Reconstructed mesh (c) Resulting depth

Figure 8: Exemple of Phase 1 execution over timestamp 133468485003417754

4.1 Phase 1: Environment Reconstruction

Due to the fact that our system is designed for use and deployment in a real-world setting, the initial phase of our approach involves the creation of a dataset suitable for subsequent stages of analysis and model development. To construct this dataset, we began by utilizing the scan mode of the HoloLens to perform a 3D reconstruction, which was later refined using the Poisson Surface Reconstruction algorithm [12]. This reconstructed mesh was then used in conjunction with ray tracing to generate depth images for each camera pose.

The results of the different sub-phases comprising the environment reconstruction phase are illustrated in Figure 8. Specifically, Figure 8a displays the image with timestamp 133468485003417754, which was captured using the HoloLens device. This image represents the initial visual input for the reconstruction process. In Figure 8b, the reconstructed mesh corresponding to the same scene depicted in Figure 8a is shown. This mesh is generated upon the completion of Phase 1.2 and represents the fully reconstructed geometry of the environment. Finally, Figure 8c illustrates the depth image derived from the ray-tracing procedure applied to the original image 133468485003417754.png. This depth map provides a pixel-wise representation of the distance from the camera to the observed scene. Additional examples demonstrating the outcomes of Phase 1 are presented in Figure 9, which showcases the process applied to two more instances. These examples further highlight the robustness and consistency of the reconstruction methodology across varying inputs.

4.1.1 Phase 1.1: Scanning the Environment

The making of the dataset started by scanning the environment room C60(CNR building, Area della Ricerca in Pisa, Institute of Information Science and Technologies), using the HoloLens 2.

Preparation for Scanning To make a proper environment scan, some preparations were necessary; we ensured that the room had adequate lighting and clear lines of sight to all surfaces. Then any unnecessary clutter that might interfere with the scan was removed.

Starting the Scan The second step was to stand in a location where we had a good initial overview of the room. The room was scanned by walking slowly around

the space, allowing the HoloLens 2 to capture and map surfaces. The depth sensors and cameras automatically detect and generate a 3D mesh of the surroundings. As you move, the HoloLens 2 displays the scan progress in real time via holographic visuals.

Completing and Saving the Scan Once the entire room has been captured, we reviewed the 3D model to ensure completeness. If necessary, we rescanned the areas that may have been missed or inaccurately captured. Finally, the data were saved to the device for further processing.

4.1.2 Phase 1.2: Environment Reconstruction

The data processing code provided by the HoloLens 2 transforms raw scanning data into a series of point clouds, each representing a portion of the scanned environment or object. These individual point clouds were then combined into a single, unified point cloud using MeshLab [2] and the Poisson Surface Reconstruction algorithm [12]. Poisson Surface Reconstruction is a widely adopted method for generating a continuous 3D mesh from discrete point cloud data. This method operates by computing an implicit function that defines the surface, creating a smooth, continuous representation from the scattered points in the cloud. In their foundational work [12], Kazhdan et al. demonstrated that surface reconstruction could be approached as a Poisson problem. This approach seeks to compute an indicator function that best aligns with the observed data points, which are often noisy and non-uniform due to real-world scanning conditions. By formulating surface reconstruction in this way, the Poisson method effectively recovers fine details and can handle imperfections in the data, making it robust for real-world applications. Unlike traditional, local surface reconstruction techniques that rely solely on nearby points to approximate surface properties, Poisson Surface Reconstruction takes into account the entire point cloud's global distribution. This global approach allows it to create surfaces that are more coherent and watertight, producing a single, continuous mesh that is free from small gaps or inconsistencies. Such coherence is especially useful in applications requiring high-quality, seamless meshes, like virtual reality modeling, digital preservation, and precise engineering simulations. The result is a highly accurate and visually consistent 3D model that can retain intricate details even from imperfectly scanned point clouds. This makes Poisson Surface Reconstruction an invaluable tool for generating professional-grade 3D assets, supporting workflows that involve digital replicas of physical objects, environments, or even complex industrial components. Using this technique in combination with HoloLens 2's spatial mapping capabilities and MeshLab's processing tools provides a powerful solution for creating immersive, accurate digital models directly from the real world.

4.1.3 Phase 1.3: Calculate Depth Images

Even though, as explained in subsection 2.3, the HoloLens provides a series of RGB images that are later used by HoloLens system to generate a mesh and depth images, the resolution of these images and the resulting mesh is relatively low. In

other words, the main issues preventing the direct use of the data produced by the HoloLens are that:

1. The resolution of the RGB images processed by HoloLens is (320x288)
2. The resolution of the depth images produced by HoloLens is (320x288)
3. The mesh produced using such low resolution images has a lot of its details lost.

To address these limitations, the dataset was reconstructed at a higher resolution, using high-resolution images (1080x1920), the original images captured by the HoloLens system, along with intrinsic and extrinsic camera data.

To transform the data to a suitable format, different steps have been followed. Starting from the high-resolution RGB images and the data of intrinsics and extrinsics for each image, we performed a reconstruction and transformation of the coordinates of the specific image to later use it to shoot the rays and get the depth results.
The depth reconstruction process is outlined in the pseudo-code provided in Algo-

Algorithm 1 Depth image extraction

```
1: for image in dataset do
2:     // from pixel to image space
3:     Read image intrinsic and extrinsic
4:     for each pixel of the image do
5:         Unproject 2D pixel(x,y) to 3D image space (x,y,z)
6:     end for
7:     // from image space to world space
8:     for each image space coordinate do
9:         cameraHomogeneous = (x, y, z, 1)
10:        worldHomogeneous = extrinsic * cameraHomogeneous
11:        (X,Y,Z) = worldHomogeneous/worldHomogeneous(3)
12:    end for
13:    Raytracing to find the distance between the origin and the mesh
14:    Save depth images
15: end for
```

rithm 1. This analysis will clarify the key segments of the code.

After loading the intrinsic and extrinsic parameters for each image, the process begins by generating a matrix of 2D coordinates representing the image space, where each cell corresponds to a specific (x, y) coordinate; this matrix has dimensions matching the image's height and width.

Lines 4 to 6 in Algorithm 1 specify the generation of **image space coordinates**. Each cell in this matrix undergoes an unprojection transformation, which involves using the intrinsic parameters to map the cell indices to coordinates in image space. Since the value of z is unknown, it is estimated by selecting a value between $[-1, 1]$ to determine the correct orientation in 3D space.

The transformation of the (x, y) indices to local (camera) space coordinates is done mathematically using the formulas in Equation 19

$$\begin{aligned} X &= \left(\frac{x - c_x}{f_x}\right) \cdot z\,, \\ Y &= \left(\frac{y - c_y}{f_y}\right) \cdot z\,, \end{aligned} \tag{19}$$

where X and Y are the resulting local space coordinates, while x and y denote the pixel indices in image space. The values c_x and c_y represent the principal point coordinates along the x- and y-axes, and f_x and f_y denote the focal lengths in those respective directions.

After obtaining the local coordinates, the next task is to convert these **image space coordinates to world coordinates**, as detailed in lines 8 to 12 of Algorithm 1. To make this transformation, the local coordinates are first converted to homogeneous coordinates. The extrinsic matrix of the image is then multiplied with these homogeneous coordinates, mapping the point from camera space to world space. Finally, the resulting homogeneous vector is normalized by dividing its first three components by the fourth, thus yielding the world coordinates. This process can be mathematically expressed as in Equation 20

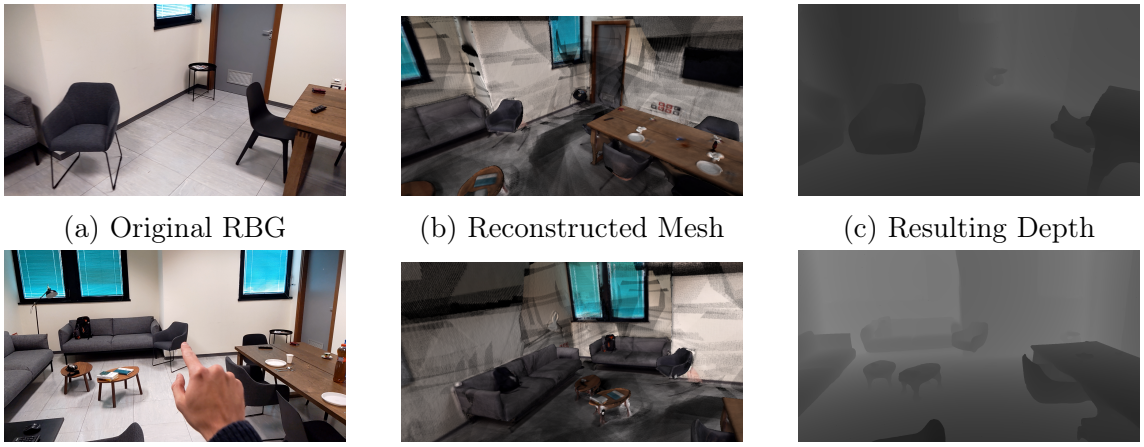
$$\begin{aligned} \text{cameraHomogeneous} &= (x, y, z, 1) \\ \text{worldHomogeneous} &= \text{extrinsic} \cdot \text{cameraHomogeneous} \\ X &= \frac{\text{worldHomogeneous}(0)}{\text{worldHomogeneous}(3)} \\ Y &= \frac{\text{worldHomogeneous}(1)}{\text{worldHomogeneous}(3)} \\ Z &= \frac{\text{worldHomogeneous}(2)}{\text{worldHomogeneous}(3)} \\ \text{worldCoords} &= (X, Y, Z) \end{aligned} \quad , \tag{20}$$

where x, y, and z are the camera space coordinates, `extrinsic` is the extrinsic matrix associated with the specific image, and X, Y, and Z are the transformed coordinates in world space.

The final step required to convert RGB images to depth images involves **ray-tracing using Embree** (line 13 in Algorithm 1). This process calculates the distance between the reconstructed world coordinates and the mesh. From each origin point in world space (x, y, z), a ray is cast in a specified direction, which is the reconstructed image in world space. For each cell in the matrix, a ray is shot, which will either return an infinite value if no intersection occurs with any face of the mesh, or a distance value (in meters) if an intersection is detected. The calculated distances are stored in a matrix, normalized within the range $[0, 255]$, and subsequently converted into a 16-bit grayscale image.

During this phase, we extracted a critical piece of information that will later be utilized by Algorithm 3. This algorithm leverages depth images and their pixel values

to perform a visibility check, ensuring that features are projected onto the correct vertices, as formalized in subsubsection 4.2.3.
The decision to employ ray tracing for depth calculation was driven by the absence of an efficient method to compute the distance between the origin point of each image pixel and its corresponding position on the mesh. Ray tracing addresses this challenge by providing a robust mechanism for determining these distances. Furthermore, the use of ray tracing enabled us to parallelize the computation, significantly improving efficiency. This approach allowed for the generation of high-quality depth information within an acceptable computational timeframe.



(a) Original RBG (b) Reconstructed Mesh (c) Resulting Depth

(d) Original RBG (e) Reconstructed Mesh (f) Resulting Depth

Figure 9: Another example of the phase 1 execution

4.2 Phase 2: Feature extraction

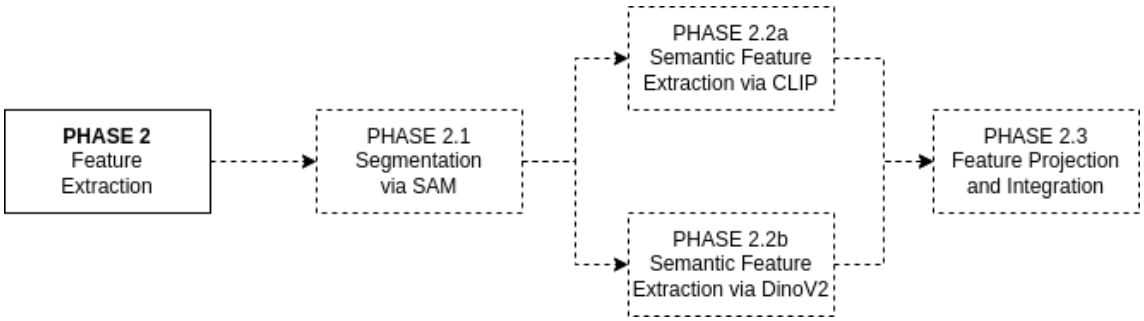
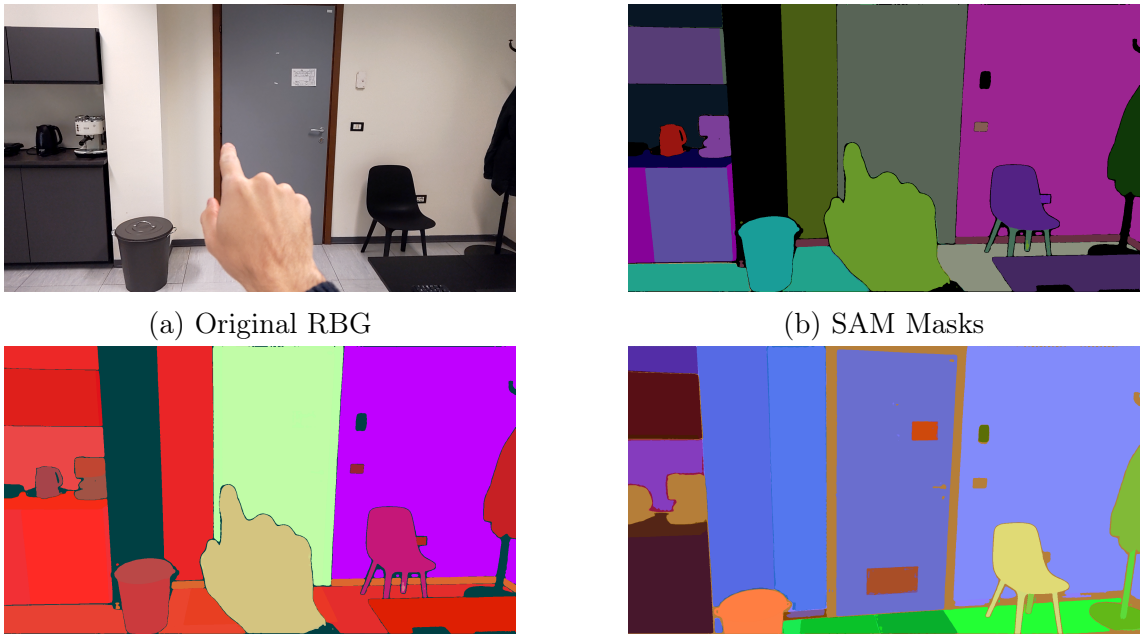


Figure 10: The workflow phase 2 with technologies

After reconstructing all the depths and properly preparing the data, the next step in preparing the dataset was extracting the features. To perform such a process, three models were used: Segment Anything [14] whose functioning is explained in subsubsection 2.2.1; OpenClip [18] whose functioning is explained in subsubsection 2.2.2 and DinoV2 [19] whose functioning is explained in subsubsection 2.2.3.

As shown in Figure 10 the steps of this phase start by segmenting the images, than the segmented masks are processed to do a semantic feature extraction us-



(a) Original RBG

(b) SAM Masks

(c) CLIP Features

(d) DinoV2 Features

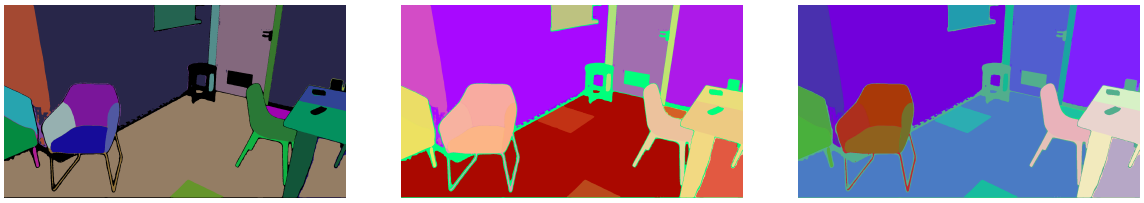
Figure 11: Example of Phase 2 execution over timestamp 133468485003417754, where the colored masks and features are obtained making a PCA reduction, of the 1024 features to 3 dimensions, later mapped in RGB

ing either CLIP [18] or DinoV2 [19] to than perform sum and normalization of the features and a final projection of the values on the mesh. In Algorithm 2 and Algorithm 3 this process has been formalized.

During the feature extraction phase, we will extract features using the masks generated by SAM, as detailed in Algorithm 2. The decision to utilize SAM-generated masks is motivated by their ability to isolate specific objects or regions of interest within an image. This targeted segmentation ensures that feature extraction focuses solely on the most meaningful and relevant portions of the image, effectively excluding irrelevant or background information.
By concentrating on these segmented regions, the resulting features exhibit reduced noise, as extraneous elements that could negatively impact the feature representation are omitted. Furthermore, this approach enhances the discriminative power of the features, enabling better differentiation between distinct objects or regions. Such improved feature quality is essential for downstream tasks that require high levels of precision and accuracy, such as object detection, classification, or instance segmentation.

Figure 11 provides an illustrative example of the application of SAM, CLIP, and DinoV2 models. In this figure, Figure 11a displays the original input data, which serves as the basis for segmentation performed by the SAM model. The masks extracted by SAM are presented in Figure 11b. These masks are subsequently processed by two distinct models. When fed into the OpenCLIP model, the results are

shown in Figure 11c. Similarly, when processed using the DinoV2 model, the output is depicted in Figure 11d. In Figure 11b, Figure 11c, and Figure 11d, areas sharing the same color correspond to the same features identified by the models. Both CLIP and DinoV2 demonstrate their ability to effectively recognize and categorize features; however, DinoV2 exhibits a superior capacity to delineate and separate distinct regions corresponding to features. This capability is particularly evident in instances where CLIP merges multiple masks into a single region, while DinoV2 retains greater granularity in distinguishing the constituent areas. Figure 12 provides additional examples of the Phase 2 process. In these examples, the input RGB images are presented in Figure 9a and Figure 9d. The corresponding outputs for each input are as follows: for the RGB image in Figure 9a, the resulting segmentation mask generated by the SAM model is shown in Figure 12a, the OpenCLIP output is depicted in Figure 12b, and the DinoV2 result is illustrated in Figure 12c. Similarly, for the RGB image in Figure 9d, the resulting SAM-generated mask is shown in Figure 12d, while the outputs from OpenCLIP and DinoV2 are presented in Figure 12e and Figure 12f, respectively. These examples further demonstrate the robustness of the models in identifying and segmenting features from diverse input data, while highlighting the differences in the granularity and precision of the outputs produced by OpenCLIP and DinoV2.



(a) SAM Masks (b) CLIP Features (c) DinoV2 Features

(d) SAM Masks (e) CLIP Features (f) DinoV2 Features

Figure 12: Another example of the phase 2 execution, the RGB are the same as Figure 9a resulting in Figure 12a, Figure 12b, Figure 12c, and Figure 9d resulting in Figure 12d, Figure 12e, Figure 12f

4.2.1 Phase 2.1: Agnostic Segmentation

Image segmentation is a crucial step in our workflow as it enables the precise partitioning of an image into distinct regions, each representing meaningful components within the visual data. To perform such partitioning, the use of the Facebook Research Group model "Segment Anything" [14] has been employed.

The segmentation process using SAM works as follows:

Algorithm 2 Feature Extraction Process

```
Load RGB images
//Extract SAM masks
Initialize SAM Model
for each image in dataset do
    Read image
    Generate masks, of the RGB images, using SAM model
    Save mask
end for
//Compute Pixel-Aligned Features
Initialize OpenCLIP Model or DinoV2 Model
for each image in dataset do
    Load saved mask
    global_feat = Extract global CLIP/DinoV2 feature from the RGB image
    for mask_segment in current image masks do
        local_feat = Extract CLIP/DinoV2 feature from the mask_segment
        roi_score = cosineSimilarity(global_feat, local_feat)
    end for
    Normalize resulting feature and store
end for
//Compute Similarity Scores:
Concatenate all Region of Interest (ROI) scores
Pass concatenated scores through softmax function to obtain a distribution
//Save Feature Maps:
for each RGB and depth image do
    Using the similarity scores, build a feature map where, for each pixel of the image, we assign the extracted feature relative to the pixel
    Save feature map
end for
```

Load the data, line 1 in Algorithm 2: The code loads all the necessary data to be feed to the models such as the images in RGB, the `configs.yaml`, and the paths to save the result.

Load and configure SAM model, line 3 in Algorithm 2 The code loads a Segment Anything Model (SAM) from the `sam_model_registry`, using a pre-trained checkpoint provided, because it simplifies the process of generating masks without needing domain-specific customizations. The SAM model is then moved to the specified device (e.g., GPU).

Generate SAM masks, lines 4 to 8 in Algorithm 2 Once the SAM model is initialized, it is used on each image in the dataset to perform the segmentation and generate the masks. Those masks are than saved in order to perform further computation later on.

4.2.2 Phase 2.2: Semantic Feature Extraction

Semantic Feature Extraction is a process in computer vision and image analysis that focuses on identifying and extracting high-level, meaningful features from images or other types of visual data. Unlike traditional feature extraction, which may involve low-level characteristics like edges or textures, semantic feature extraction emphasizes contextually rich information that corresponds to objects, scenes, or relationships within the image. This process is crucial in applications where understanding the "meaning" or "semantics" of the visual data is essential. In our system, we have two different models that can perform the feature extraction phase, OpenCLIP [18] and DinoV2 [19], where both are very capable models, and we want to compare their performance and capabilities.
The whole process to extract the features goes as follows:

Load and initialize the OpenCLIP/DinoV2 model, line 10 in Algorithm 2 The OpenCLIP/DinoV2 model (a variant of CLIP, or the DinoV2, which is used for zero-shot learning tasks) is loaded and initialized with pre-trained weights. The model is moved to the GPU (`model.cuda()`), and the evaluation mode (`model.eval()`) is set, meaning the model won't update its weights and will only perform inference.

Compute pixel-aligned features, lines 11 to 19 in Algorithm 2 The code iterates over the dataset again, but this time it processes the previously saved mask files. For each image, the corresponding masks are loaded. The code extracts global CLIP/DinoV2 features for the whole image by preprocessing it (resizing, normalization) and passing it through the CLIP model (`model.encode_image`). The resulting feature is normalized and stored as `global_feat`. This `global_feat` is than compared via cosine similarity to the feature that is extracted for each mask of the image `local_feat` calculating a ROI or Region Of Interest.

Compute similarity scores, lines 20 to 22 in Algorithm 2 The similarity scores for all ROIs are concatenated and passed through a softmax function to

convert the scores into a distribution. Finally, an output feature map (`outfeat`) is initialized with the dimensions of the original image, where each pixel is aligned with its corresponding feature, making it pixel-aligned.

Save all feature maps, lines 23 to 27 in Algorithm 2 For each RGB and depth, a feature map is saved.

4.2.3 Phase 2.3: Feature Projection and Integration

The final step before querying the scene involves projecting the features in the mesh. To do so, we had to perform a 3D-to-2D projection of mesh vertices, transforming their coordinates from world space to image space and comparing their projected depth to a depth image for further analysis. We utilized the libraries VCG (Visual Computing Library) for handling 3D meshes, Eigen for matrix operations, and OpenCV for working with image data.

Algorithm 3 Vertex Projection

1: `Retrieve 3D Vertex`
2: `Convert to Homogeneous Coordinates`
3: `Transform to Camera Coordinates`
4: `Visibility Check`
5: `Project to Image Plane`
6: **if** projected pixel (x, y) lies within the depth image bounds **then**
7: Compare the depth value from the depth image with the z coordinate of the vertex in camera space, representing the vertex's distance from the camera.
8: **if** depth check is successful **then**
9: Sum the projected 2D pixel feature with the feature of the vertex and normalize accordingly.
10: **end if**
11: **end if**

The steps we followed to achieve such a projection are:

- Line 1, Algorithm 3: Retrieves the 3D position of a vertex from the mesh by its index
- Line 2, Algorithm 3: The vertex is converted from a 3D point to homogeneous coordinates by adding an extra component 1.0. This is necessary for matrix multiplication in camera transformations.
- Line 3, Algorithm 3: The vertex is transformed from world coordinates to camera coordinates by multiplying it with the inverse of the extrinsic matrix. This transformation accounts for the camera's position and orientation in the scene. The resulting coordinates are normalized by dividing by the homogeneous coordinate *(camCoords(3))*, bringing it back to 3D space.
- Line 4, Algorithm 3: A control to check if the vertex is in front of the camera, by evaluating the z value in camera coordinates), is performed. If it's negative

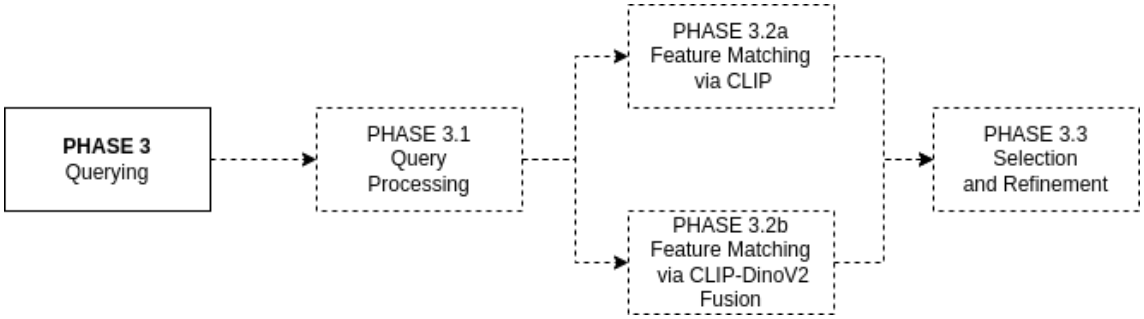


Figure 13: The workflow phase 3 with technologies

or zero, the point is either behind or at the same position as the camera, and it won't be projected.

- Line 5, Algorithm 3: The 3D point is projected onto the 2D image plane using the camera's intrinsic matrix, which describes the internal parameters of the camera.
- Line 6, Algorithm 3: If the projected pixel *(x, y)* lies within the bounds of the depth image. If the pixel is out of bounds, it is discarded.
- Line 7, Algorithm 3: The code compares the depth value from the depth image with the camera coordinate's z value, which represents the distance of the vertex in camera space. A threshold is applied to allow some tolerance. If the depth in the image is greater than or equal to the vertex's depth (plus the threshold), it means the vertex lies within a valid range.
- Lines 8-9, Algorithm 3: If the depth check is successful, the projected 2D-pixel feature is summed and normalized over the feature of the vertex.

The sum and normalization are executed, summing the vectors of 1024 features from different picture pixels whose projection points to the same vector. The normalization is a simple L2-normalization.

4.3 Phase 3: Querying

The final phase of our system, once the environment has been reconstructed and the features have been properly attached to the mesh, is the querying. In other words, once Phase 1 and Phase 2 have been performed, we are ready to ask questions to our system. Regardless of whether CLIP or DinoV2 is used as the model for querying, the initial step involves transforming the query into a format compatible with the model's input requirements. Once this transformation is complete, similarity metrics can be applied to determine which parts of the mesh correspond to the query. These identified sections are then selected, highlighted, and returned as the result.

4.3.1 Phase 3.1: Query Processing

This phase consists of the process of transforming natural language queries into numerical representations using the openCLIP model, an open-source adaptation of

Algorithm 4 Prompt to feature

```
query = Initialize a query
model = Initialize OpenCLIP Model using the same pretrained set
from the feature extractor in Algorithm 2
query = Tokenize the query
query = model(query) Use CLIP to encode the text
query = Normalize the CLIP feature
Save the result
```

the CLIP (Contrastive Language-Image Pretraining) model. It is structured to take user-defined questions or prompts, convert them into machine-understandable vectors, and store these vectors as binary files. These vectors, or "features," can later be used for tasks like similarity search, where the system compares these precomputed features with other data to find matches or retrieve relevant information.
For each prompt, the following steps are taken:

- **Tokenization and Encoding:** The prompt text is fed through the tokenizer, and the resulting tokens are passed into the model. This generates a "text embedding" for the prompt — a high-dimensional vector that represents the semantic meaning of the text in a form the model understands.
- **Normalization:** The text embedding is then normalized, adjusting the vector so that it has a consistent length. This step is important for similarity comparisons, as normalized vectors can be compared more easily and meaningfully.
- **Saving the Encoded Feature:** After normalization, the values are then saved.

While the DinoV2 model demonstrates promising capabilities, it currently lacks an automated mechanism to transform text queries into a format compatible with similarity score calculations. This limitation restricts the ability to perform direct textual queries on the model.

4.3.2 Phase 3.2a: Feature Matching via CLIP

Algorithm 5 Feature Querying Using CLIP

```
query = load feature query generated in Algorithm 4
features = load all features computed in Algorithm 3
score = calculate similarity_metric(features, query)
Map similarity score to the color ramp and apply to environment
visualization
```

This phase involves comparing the features computed across the data processing pipeline with the feature query generated in the preceding step. The objective is to assess whether any part of the environment matches the input query based on a similarity measure. The phase is structured as follows:

1. Depth Calculation (Algorithm 1): Depth values are computed, providing foundational data to establish a three-dimensional spatial structure for further feature processing.

2. Feature Extraction (Algorithm 2): Specific features are extracted from the data, facilitating the identification of unique characteristics within the environment.

3. Vertex Projection (Algorithm 3): These features are then re-projected onto the corresponding vertices in the environment, aligning them with their spatial context.

4. Feature Query Calculation (Algorithm 4): The query feature is transformed to match the spatial and feature space of the environment, enabling an effective comparison.

After these steps, the query feature, now aligned with the environment's feature space, is compared to the extracted features. The similarity between the query and the environment is assessed using a suitable metric, such as Euclidean distance, cosine similarity, or Manhattan distance. This comparison determines the degree to which any portion of the environment resembles the query.

4.3.3 Phase 3.2b: Feature Matching via CLIP-DinoV2 Fusion

While the DinoV2 model demonstrates promising capabilities, it currently lacks an automated mechanism to transform text queries into a format compatible with similarity score calculations. This limitation restricts the ability to perform direct textual queries on the model. However, an alternative approach exists: we can select a specific vertex feature and utilize it as a query, as shown in Algorithm 6.
This functionality enables the creation of an interactive, click-based query system. By selecting a particular point on the model (e.g., clicking on a chair), the model can identify and highlight similar features within the environment (e.g., all seats across the environment). Such a system facilitates spatially contextual queries and interactive exploration of feature similarities.
While this potential is recognized, the current implementation does not explore or evaluate its full functionality.

Algorithm 6 Feature Querying Using DinoV2

```
query = select a feature by its vertex ID
features = load all features computed in Algorithm 3
score = calculate similarity_metric(features, query)
Map similarity score to color ramp and apply to environment
visualization
```

After analyzing the query results from both the CLIP and DinoV2 models, we developed a method to combine these outputs for a more robust feature-matching process. The fusion approach follows these key steps:

1. Transform the text query into a feature query using the CLIP model.
2. Perform a similarity matching using the CLIP model, obtaining initial similarity scores across the environment.
3. Identify the highest similarity scores from the CLIP output.
4. Using the vertices with the highest scores, apply the DinoV2 model for a second round of feature matching to refine and validate the results.

This two-stage approach leverages the strengths of both models: CLIP enables text-to-feature mapping, while DinoV2 provides enhanced spatial feature matching. The final output is based on a combination of similarity scores, where DinoV2 scores replace CLIP scores if they indicate a higher match at corresponding indices.

Algorithm 7 Feature Querying by Fusing CLIP and DinoV2

```
query = Extract feature query as per Algorithm 4
clip_features = Load all features computed by CLIP (Algorithm 3)
similarity_scores = similarity_metric(clip_features, query)
for value in similarity_scores do
    if value == max(similarity_scores) then      ▷ There may be multiple maximum values
        dino_features = Load all features computed by DinoV2 (Algorithm 3)
        dino_query = Select the index of the corresponding vertex with maximum similarity
        dino_scores = similarity_metric(dino_features, dino_query)
        for idx in dino_scores do
            if dino_scores[idx] > similarity_scores[idx] then
                similarity_scores[idx] = dino_scores[idx]
            end if
        end for
    end if
end for
Map similarity_scores to a color ramp and apply to environment visualization
```

4.3.4 Phase 3.3: Selection and Refinement

Once the similarity scores are computed — whether using the CLIP model or the CLIP-DINOv2 fusion — the final step in the querying process involves both selecting the most relevant values and visualizing the results. The selection step identifies the vertices with the highest similarity scores, representing the elements most likely to address the query. To further enhance interpretability, these selected vertices are visualized within the MeshLab [2] environment.

For the visualization process, a color ramp inspired by the Pyplot "jet" colormap is employed. This color ramp encodes similarity values as follows: regions with

high similarity scores (closer matches to the query) are represented in red, areas with lower similarity are depicted in green, and regions with missing or incomplete data are marked in blue. This approach allows for an intuitive and visually distinct representation of similarity across the dataset.

Chapter 5 Experiments and Results

Following the introduction of our proposed method, this chapter provides a detailed explanation of the implementation, including an in-depth discussion of the specific models utilized. Furthermore, it offers an analysis of the results obtained from executing natural language queries within the system.

To reiterate, the primary objective of the system is to develop a framework capable of integrating semantic information into the vertices of a 3D reconstruction mesh (Figure 14), generated from a scanned real-world environment. This semantic enrichment enables the system to process and respond to natural language queries with zero-shot capabilities, demonstrating its effectiveness and versatility in interacting with the reconstructed environment.

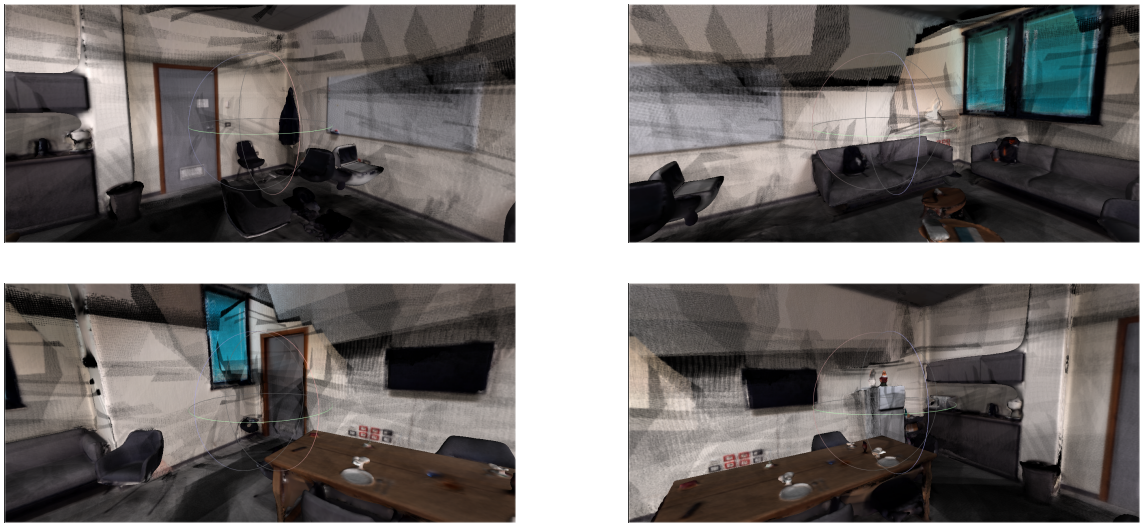


Figure 14: Overview of the reconstructed CNR-C60 room

5.1 Implementation details

Before actually Selecting and running the queries, that will be used to test our system, lets introduce some implementation details to clarify how our approach, presented in section 4 went from theory to practice.

Calculate Depth Images, Algorithm 1: The procedures described in this section were implemented in C++ to leverage the `EmbreeAdaptor` [5] from `VCGlib` [3], enabling efficient use of Intel Embree [26] for ray tracing.

Efficient generation of depth images was achieved by employing OpenMP for parallelization, as the generation process for each image is independent of the others. Once the depth images are generated for every RGB image, several auxiliary files are created to facilitate subsequent processing. Among these is a file containing the

intrinsic camera parameters for each image, specifically the values of f_x, f_y, c_x, and c_y.

Additionally, an `association.txt` file is generated to link each depth image with its corresponding RGB image. The `odometry.gt.sim` file is also produced by deriving the extrinsic parameters from the `*_pv.txt` files and multiplying these by the inverse of the fixed extrinsics provided in `Depth Long Throw_extrinsics.txt`. This results in a `odometry.gt.sim` file containing a 3×4 matrix for each image, which represents the camera pose as:

$$\text{poses} = \text{extrinsic} \cdot \text{fixed\_extrinsics}^{-1}, \tag{21}$$

where `extrinsic` denotes the relative extrinsic matrix derived from the image's `*_pv.txt` file, and `fixed_extrinsics` refers to the fixed extrinsic parameters obtained from `Depth Long Throw_extrinsics.txt`.

Finally, two directories, `rgb/` and `depth/`, are created to store the RGB and depth images, respectively.

Agnostic Segmentation (Algorithm 2): The agnostic segmentation process is initiated by employing a mask generator, specifically the `SamAutomaticMaskGenerator`, which is configured to generate masks from the input images. The generator is initialized with various parameters, such as `points_per_side`, `pred_iou_thresh` (Intersection over Union threshold), and cropping specifications, to ensure accurate segmentation. The pretrained model utilized in this context is `sam_vit_h_4b8939`, which is based on the gradSLAM [15] foundation. This particular checkpoint is chosen due to the superior capability of the ViT-H (Vision Transformer, Hierarchical) backbone in capturing intricate and hierarchical image features. The ViT-H architecture has been extensively trained on diverse datasets to generalize well across a wide range of visual tasks, making it suitable for our agnostic segmentation pipeline. Once the SAM model is initialized, the script ensures that the output directory for saving masks exists. It then iterates over the dataset of input images, performing the following steps for each image:

- The image is read using OpenCV (`cv2`) and converted into RGB format.
- The SAM model generates segmentation masks for the image.
- The first mask (`masks[0]['segmentation']`) is selected for further processing.
- The generated masks are saved in `.pkl` format to the local disk. This step is necessary due to the large size of each mask, making local storage more efficient for subsequent feature extraction.

Semantic Feature Extraction (Algorithm 2): The process of semantic feature extraction involves utilizing two pretrained models: OpenCLIP and DinoV2. For OpenCLIP, the model employed is `ViT-H-14`, which was trained on the extensive `laion2b_s32b_b79k` dataset. This dataset contains a massive collection of image-text pairs, enabling the model to excel in associating semantic features with visual

content. The pretrained OpenCLIP model is particularly adept at capturing high-level semantic features, making it suitable for applications requiring robust image-text understanding.
For DinoV2, the model `facebookresearch/dinov2` was used, with the specific configuration `dinov2_vitl14`. This model was trained using a self-supervised learning approach, leveraging diverse datasets to develop an exceptional capacity for distinguishing fine-grained features within images. DinoV2 is well-suited for this task due to its ability to learn detailed representations without requiring labeled data.
Unlike other components of the pipeline, which are implemented in C++, the feature extraction process was developed in Python to leverage the robust libraries and tools available for machine learning and neural network inference. This implementation allows efficient integration of the pretrained models into the pipeline, ensuring accurate and scalable feature extraction.

5.2 Defining the Queries

The final step before conducting the actual experiments was to select specific queries to run on the newly normalized data. These queries were chosen not only to test the system's ability to identify the correct objects accurately but also to evaluate whether it can distinguish between single and multiple objects and respond effectively to both broad and detailed queries.

The process of generating an answer from a query is divided into two distinct phases:

Query Transformation In this initial phase, the system transforms the query into a format that the model can interpret. The goal is to bridge the gap between the raw query format and the structured data the model requires to process it effectively.

Data Comparison and Result Generation Once the query is in a compatible format, the model compares it against the underlying data. The output is presented as the best possible response based on the data available and the specifics of the query.

This two-phase approach ensures that the query is not only understood by the model but also matched accurately to the data, resulting in precise and relevant answers.

5.3 Running the queries

With the methodology established to process raw data into CLIP and DinoV2 features and subsequently query these features, we now proceed to analyze some of the resulting outputs. Due to the absence of a standardized scoring mechanism for quantitative evaluation, the results are primarily assessed through visual comparison.

In these evaluations, our primary objectives are as follows:

ID	Query	Expected result
1	"Where are the doors?"	The two doors should be highlighted.
2	"Is there a TV?"	The TV, located on the wall, should be highlighted.
3	"Can I find my backpack?"	The two backpacks on the couches should be highlighted.
4	"Do you see a whiteboard?"	Only the whiteboard should be highlighted.
5	"Where can I sit?"	All chairs and couches should be highlighted.
6	"Where are the couches?"	Both couches should be highlighted.
7	"Where is the left couch?"	Only the couch on the left should be highlighted.
8	"Where is the kitchen?"	The kitchen wall should be highlighted.
9	"Where can I put the trash?"	The trashcan should be highlighted.
10	"Where is the gnome?"	The gnome in the fridge should be highlighted.
11	"Is there something strange in the room?"	We expect the gnome to be highlighted
12	"Are there anythings out of place?"	We expect any out-of-place object to be selected

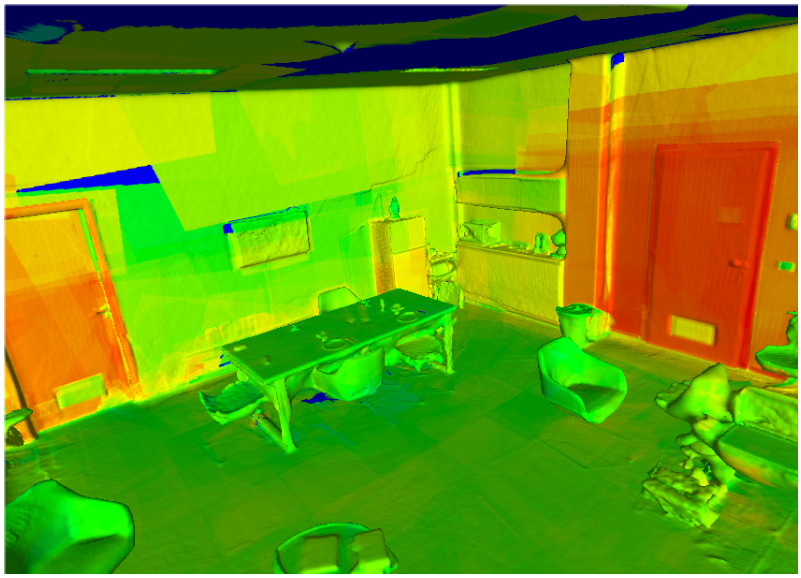
Table 2: Query table with expected results for each question.

- **Object Recognition:** Verify if the models can accurately identify the requested objects within the environment, ensuring effective alignment between the query and the environment's features.
- **Singular vs. Plural Differentiation:** Assess the models' ability to distinguish between singular and plural entities, an indicator of contextual understanding.
- **Out-of-Place Object Identification:** Observe whether the models can detect objects that are unusual or unexpected within the given setting, testing their capability for contextual awareness.
- **Positional Recognition:** Evaluate the models' capacity to recognize objects based on their spatial positioning within the environment, indicating spatial and relational understanding.

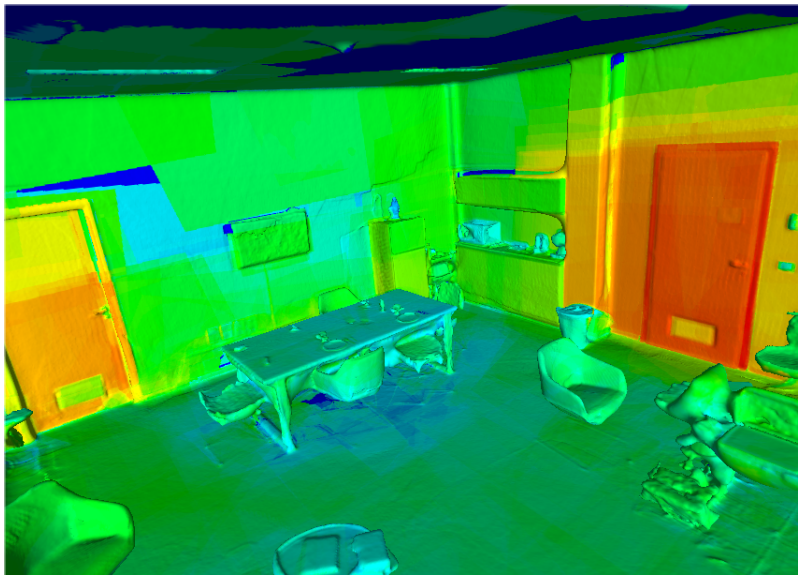
These visual comparisons serve as the basis for evaluating the effectiveness of the CLIP and DinoV2 models in handling complex querying tasks across varied contexts.

5.3.1 Where are the doors?

The first query submitted to the model is, "Where are the doors?" The room contains two separate doors, so the expectation is that the system will correctly identify and select both doors.



(a) Query 1: CLIP Result



(b) Query 1: CLIP and DinoV2 Fusion Result

Figure 15: Results for Query 1: "Where are the doors?"

For Query 1, the results are as follows:

CLIP Result (Figure 15a): In this image, the selection made by CLIP is mostly accurate, successfully identifying the doors' positions on the wall. The "reddest" areas in the selection correspond to the doors themselves, with one door being more precisely highlighted than the other. Since yellow in the color map indicates regions

that are somewhat relevant to the query, it is observed that much of the wall and areas surrounding the doors are also included as part of the response to the query.

CLIP and DinoV2 Fusion Result (Figure 15b): In the fusion result, the segmentation of the environment is more precise compared to the CLIP-only selection. Unlike the CLIP image (Figure 15a), where large portions of the wall were considered part of the response, the fusion result provides a more concise selection, specifically highlighting the doors and excluding most of the wall.

Overall, for Query 1, the model successfully identified the correct elements, thereby fulfilling the expected result.

5.3.2 Is there a TV?

The second query posed to the model was, "Is there a TV in the room?" This query was designed to assess the model's ability to distinguish between the television, laptops, whiteboard, and doors, given their visual or functional similarities. The expectation was that the model would correctly identify and select only the TV mounted on the wall. For Query 2, the results are as follows:

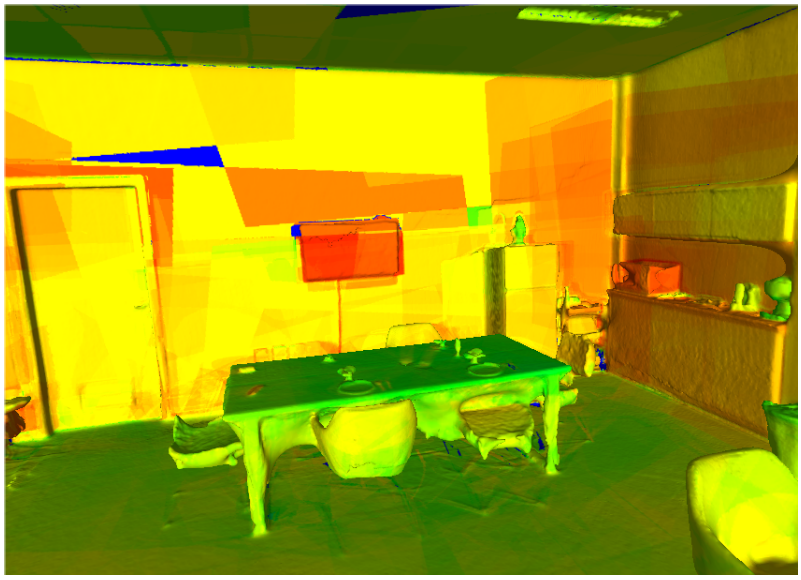
CLIP Result (Figure 16a): Using the CLIP features, the model selects most of the wall area, which it considers part of the response. While the TV is marked as the primary answer, the microwave on the counter (visible on the right side of Figure 16a) is also highlighted with a similar level of importance. This indicates that the model, with CLIP alone, struggles to completely exclude other electronic devices from the selection, resulting in a less precise outcome.

CLIP and DinoV2 Fusion Result (Figure 16b): The fusion approach yields a significantly more accurate result, where only the TV is selected. Unlike the CLIP-only selection, this result does not include the surrounding wall or other objects, such as the microwave, which are entirely ignored. Additionally, while laptops are present in the room (though not visible in the figure), they are not selected in this fusion result, demonstrating improved specificity.

Overall, this query illustrates the precision enhancement provided by the DinoV2 segmentation. In the CLIP-only selection, nearly any flat or technological object is included in the response. In contrast, the CLIP and DinoV2 fusion focuses solely on the actual target, highlighting the TV as the correct answer to the query.

5.3.3 Can I find my backpack?

For the third query, we asked the model, "Can I find my backpack?" This query is phrased as a personal question to assess whether the model can handle informal language. Additionally, this query aims to test the model's ability to distinguish the backpacks from the couches on which they are placed, without mistakenly including the entire couch. For Query 3, the results are as follows:



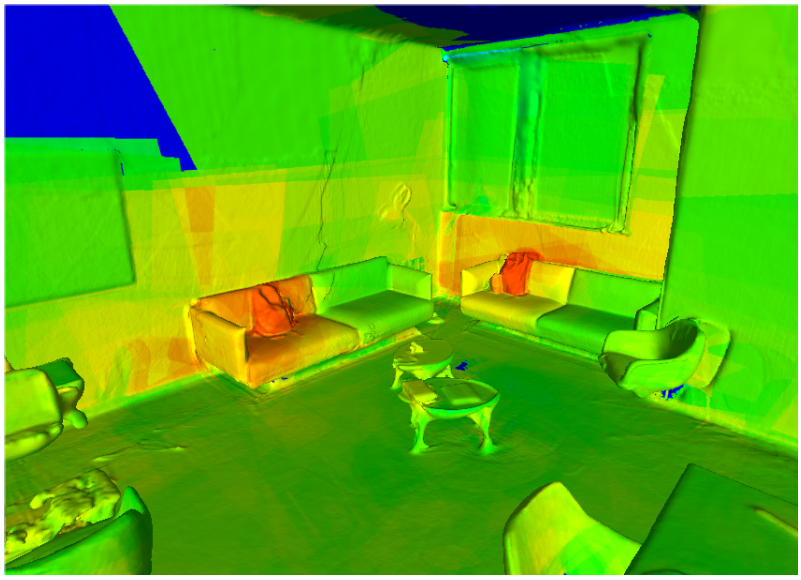
(a) Query 2: CLIP Result

(b) Query 2: CLIP and DinoV2 Fusion Result

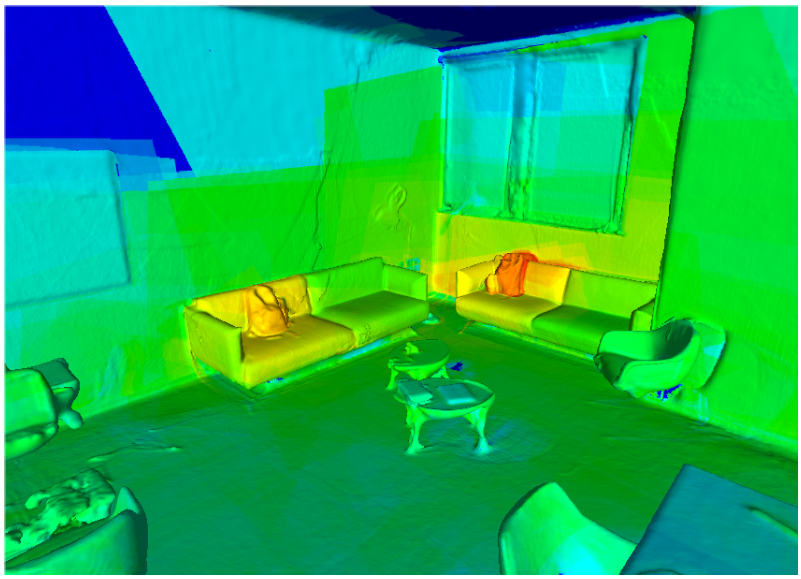
Figure 16: Results for Query 2: "Is there a TV?"

CLIP Result (Figure 17a): In this instance, the CLIP model provides a satisfactory selection. Although the backpacks are placed on the couch cushions, the model effectively identifies both backpacks. The right backpack is highlighted more accurately, while the left one includes part of the couch cushions in the selection as relevant to the query.

CLIP and DinoV2 Fusion Result (Figure 17b): With the CLIP-DinoV2 fusion, the results are similar to those of the CLIP-only selection. The primary improvement over the CLIP result is the exclusion of the wall, which is not considered part of the response.



(a) Query 3: CLIP Result



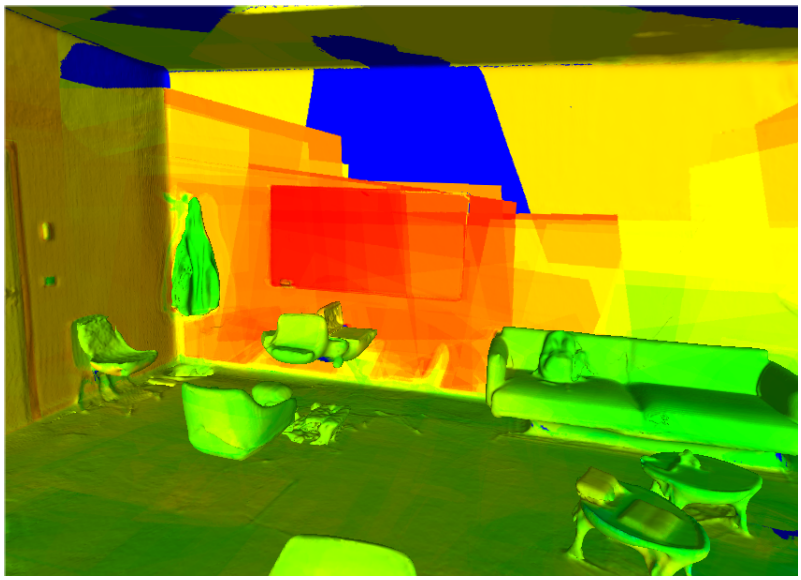
(b) Query 3: CLIP and DinoV2 Fusion Result

Figure 17: Results for Query 3: “Can I find my backpack?”

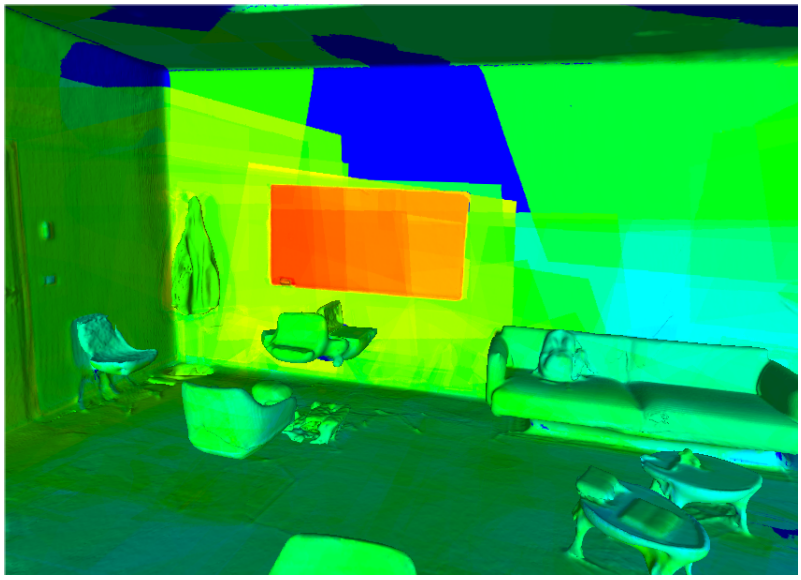
Overall, for this query, both the CLIP and CLIP-DinoV2 Fusion models yield a satisfactory result with minimal interference from unrelated elements in the selection.

5.3.4 Do you see a whiteboard?

The fourth query, “Do you see a whiteboard?”, is purposefully informal, similar to the phrasing in the third query. The objective here, as in the second query, is to assess whether the model can differentiate between objects of similar shape and placement based solely on their purpose. For Query 4, the results are as follows:



(a) Query 4: CLIP Result



(b) Query 4: CLIP and DinoV2 Fusion Result

Figure 18: Results for Query 4: “Do you see a whiteboard?”

CLIP Result (Figure 18a): The CLIP model's selection for this query is notably broad. While the whiteboard is the "reddest" area in the selection, the surrounding wall also displays a gradient from orange to yellow, indicating that these areas are considered somewhat related to the whiteboard. However, these areas are, in reality, just wall space and unrelated to the query's focus.

CLIP and DinoV2 Fusion Result (Figure 18b): In this instance, the CLIP-DinoV2 fusion demonstrates significant precision by discarding any erroneous selection of the wall or other surrounding areas and focusing exclusively on the whiteboard.

Overall, this query highlights a limitation of the CLIP approach: thin objects like the whiteboard can blend visually with nearby structures, leading to over-selection. In contrast, the CLIP-DinoV2 fusion achieves a highly accurate selection, underscoring its superior performance in scenarios requiring precise segmentation.

5.3.5 Where can I sit?

The fifth query, "Where can I sit?", is intentionally a broad request. Unlike previous queries, it does not seek a specific object but rather locations where the action of sitting can be performed. In other words, the goal is not to identify only chairs but all areas suitable for sitting. For Query 5, the results are as follows:

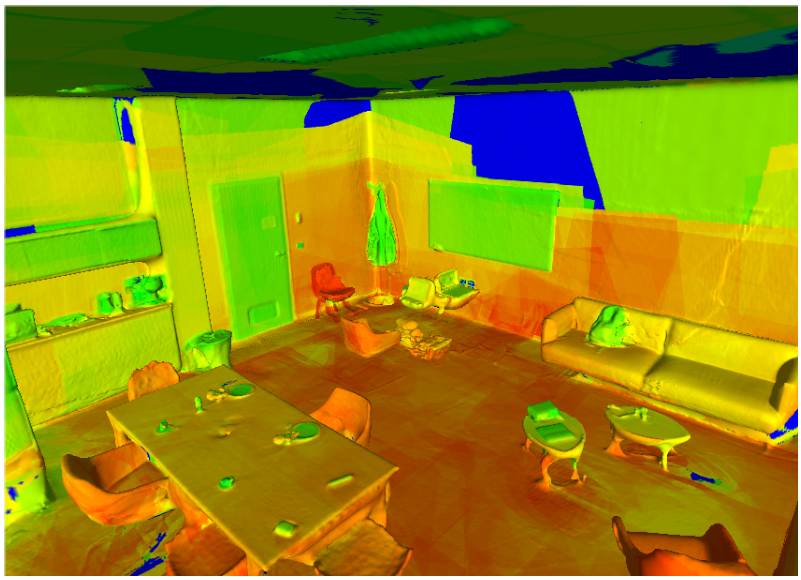
CLIP Result (Figure 19a): In this case, the CLIP model's response is very broad, which aligns with expectations given the query's general nature. Most of the chairs and couches are highlighted in varying intensities, from red to yellow, indicating their relevance to the query. However, a large portion of the floor is also selected, which is unnecessary and detracts from the clarity of the result.

CLIP and DinoV2 Fusion Result (Figure 19b): In this instance, the CLIP-DinoV2 fusion approach does not significantly improve the selection over the CLIP-only method. This outcome may be influenced by the current approach used for fusion.

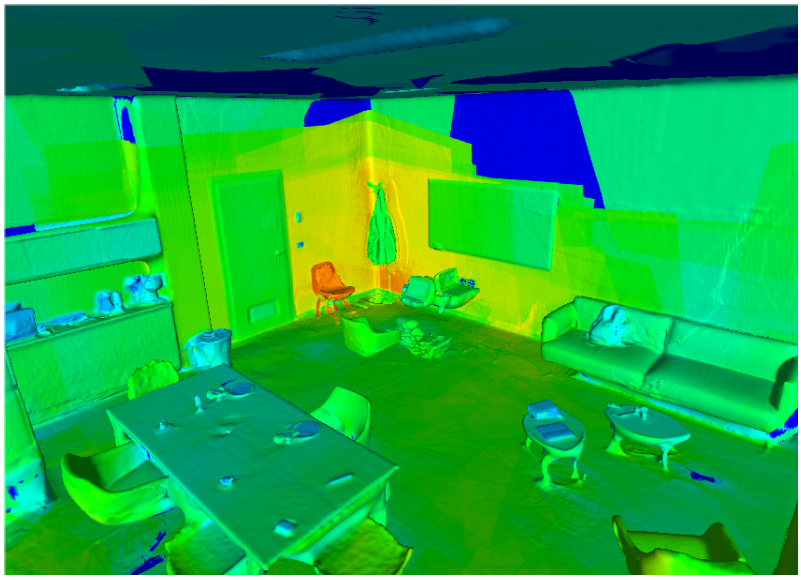
Overall, this query demonstrates that the model is capable of interpreting and addressing an action-based request, such as finding potential seating areas. However, to improve accuracy and minimize irrelevant selections, the fusion method requires further refinement for such broad requests.

5.3.6 Where are the couches?

The seventh query, "Where are the couches?", was designed to assess the model's ability to distinguish between objects with closely related purposes, such as chairs and couches. Unlike Queries 2 and 4, which focused on wall-mounted objects, this query targets items positioned in the "middle" of the room. For Query 6, the results are as follows:

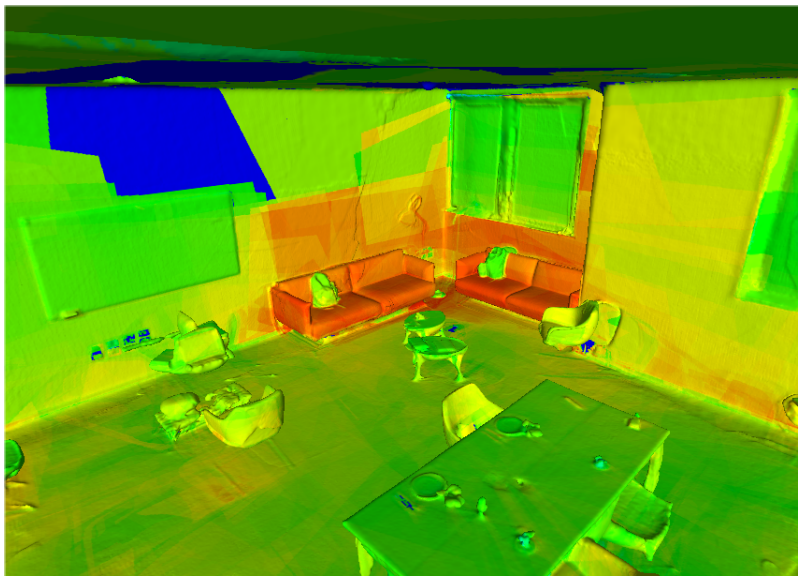


(a) Query 5: CLIP Result

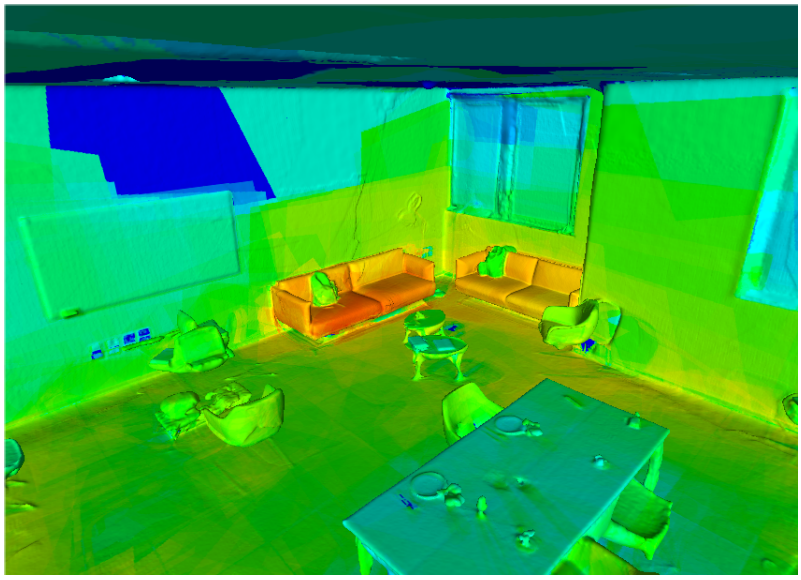


(b) Query 5: CLIP and DinoV2 Fusion Result

Figure 19: Results for Query 5: “Where can I sit?”



(a) Query 6: CLIP Result



(b) Query 6: CLIP and DinoV2 Fusion Result

Figure 20: Results for Query 6: “Where are the couches?”

CLIP Result (Figure 20a): The CLIP model effectively selects the couches as the primary response to the query, with minimal interference from other objects. The chairs are not included in the selection, and the only minor issue is that a small portion of the wall behind the couches is considered part of the answer.

CLIP and DinoV2 Fusion Result (Figure 20b): The CLIP-DinoV2 fusion approach provides an even cleaner result, selecting only the couches without including the wall or any surrounding chairs.

Overall, this query performs as expected with both the CLIP-only and CLIP-DinoV2 fusion models. Notably, the backpacks, which are present near the couches, are not selected in either result, indicating that the model can clearly segment and distinguish between the features of backpacks and couches.

5.3.7 Where is the left couch?

In the sixth query, "Where is the left couch?", we prompt the model to infer spatial information about the objects in the room. Unlike Query 6, which simply sought to locate the couches, this query focuses on testing the model's spatial understanding and its ability to identify a specific position. For Query 7, the results are as follows:

CLIP Result (Figure 21a): Unfortunately, in this case, the CLIP model does not return the expected result. Both couches are selected, with one appearing slightly "redder" than the other. However, it is unclear if this distinction corresponds to the "left" clause in the query, as no definitive preference for the left couch is observed.

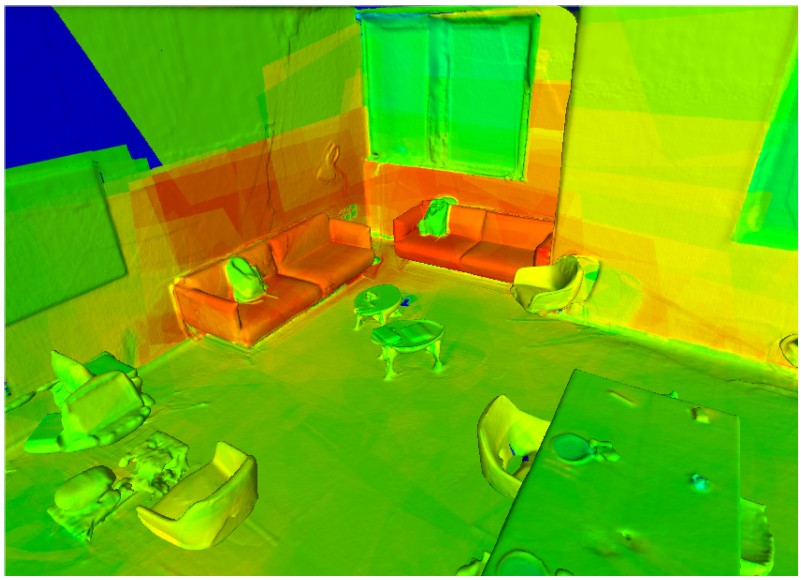
CLIP and DinoV2 Fusion Result (Figure 21b): Similar to the CLIP result, the CLIP-DinoV2 fusion method selects both couches without focusing on a single couch.

Overall, this query demonstrates that the model does not adequately handle requests involving spatial qualifiers, such as "left," and thus fails to return a significant result for queries that rely on cardinal positioning.

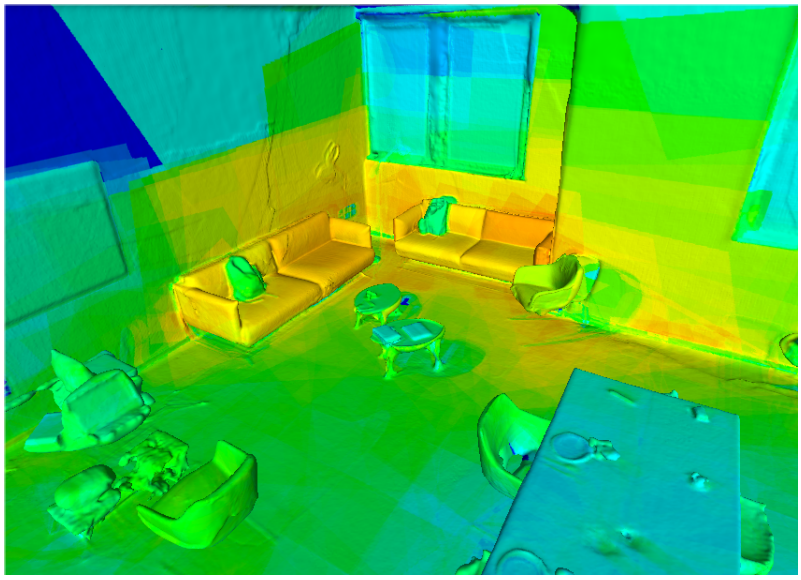
5.3.8 Where is the kitchen?

The eighth query, "Where is the kitchen?", is designed to determine if the model can identify an area based on a collection of objects rather than individual items. This query explores whether the model can understand that diverse objects, potentially with different functions, collectively define a specific area with a unified purpose. For Query 8, the results are as follows:

CLIP Result (Figure 22a): In the CLIP selection, the model identifies various objects that constitute the kitchen area, though the result is somewhat sparse. The selection appears to focus more on the walls behind the counter than on the objects

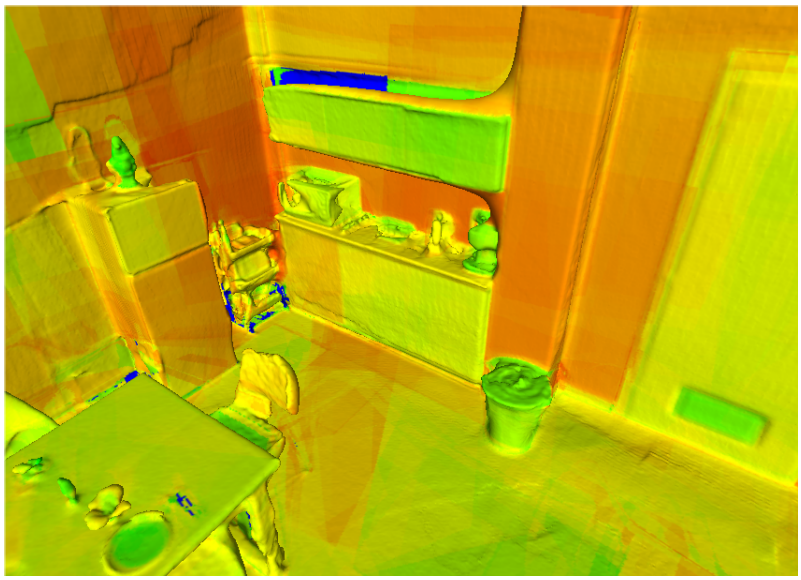


(a) Query 7: CLIP Result

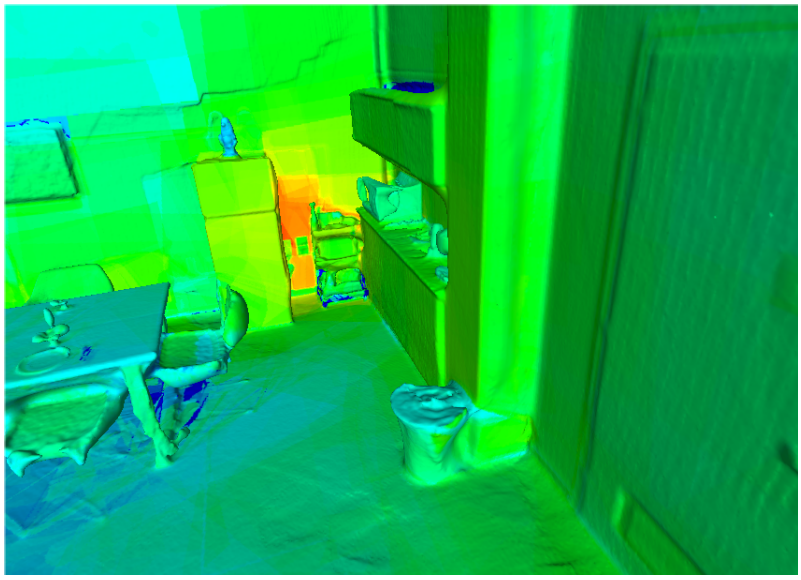


(b) Query 7: CLIP and DinoV2 Fusion Result

Figure 21: Results for Query 7: “Where is the left couch?”



(a) Query 8: CLIP Result



(b) Query 8: CLIP and DinoV2 Fusion Result

Figure 22: Results for Query 8: “Where is the kitchen?”

themselves. Despite this, the result provides a reasonably accurate indication of the kitchen's location.

CLIP and DinoV2 Fusion Result (Figure 22b): The CLIP-DinoV2 fusion result is less effective than expected, with the selection limited to a small portion of what should encompass the entire kitchen area. This limitation may stem from constraints within the fusion approach.

Overall, the eighth query yields a more satisfactory result using the CLIP method than with the CLIP-DinoV2 fusion. Although the selection is somewhat sparse, the kitchen area is generally marked correctly, making the result reasonably acceptable.

5.3.9 Where can I put the trash?

The ninth query, similar to the fifth, does not specify an object but rather an action: "Where can I put the trash?" The expected outcome is that the model identifies the trash bin in the corner, or perhaps the kitchen as an appropriate area. For Query 9, the results are as follows:

CLIP Result (Figure 23a): In the CLIP selection, the trashcan is successfully highlighted, as expected. However, the selection also includes the wall behind the trashcan, which is irrelevant to the query.

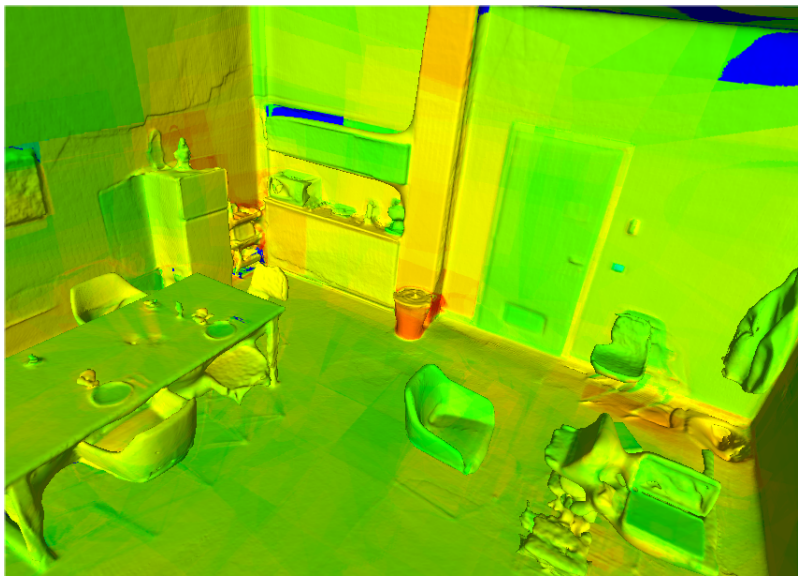
CLIP and DinoV2 Fusion Result (Figure 23b): The CLIP-DinoV2 fusion result does not identify the trashcan itself but instead selects the wall behind it. This misidentification appears to result from the feature projection erroneously associating the wall with the intended purpose.

Overall, both models are capable of identifying the general area related to the query. However, the CLIP-DinoV2 fusion fails to select the correct portion of the object, resulting in a less precise outcome.

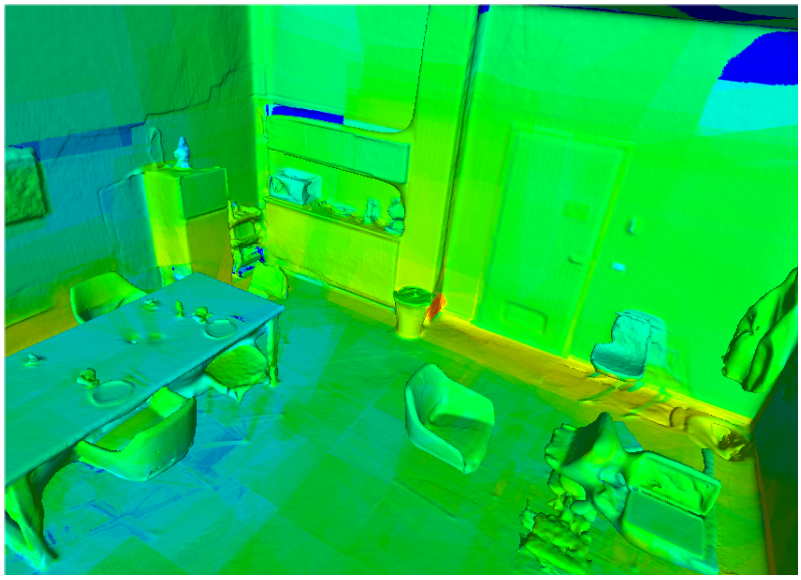
5.3.10 Where is the gnome?

The tenth query, "Where is the gnome?", was designed to locate an unusual object within the environment. The choice of a gnome statue — a typically unexpected item in an office or kitchen setting — serves to test whether the model can retain an object's "feature identity" even when it is out of context. For Query 10, the results are as follows:

CLIP Result (Figure 24a): When using the CLIP querying method, the gnome is correctly selected, but a significant portion of the wall behind it is also highlighted as relevant. This may be due to how the object is represented and the way its features are projected in relation to the surrounding area.

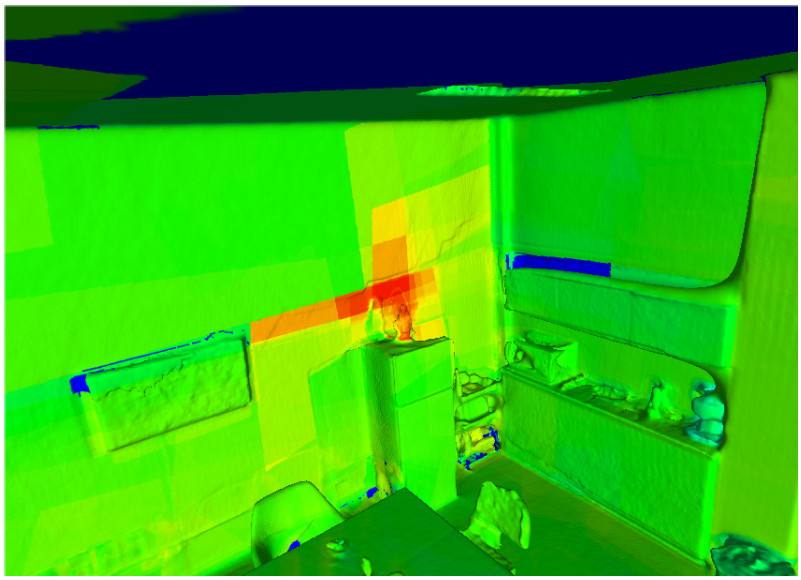


(a) Query 9: CLIP Result

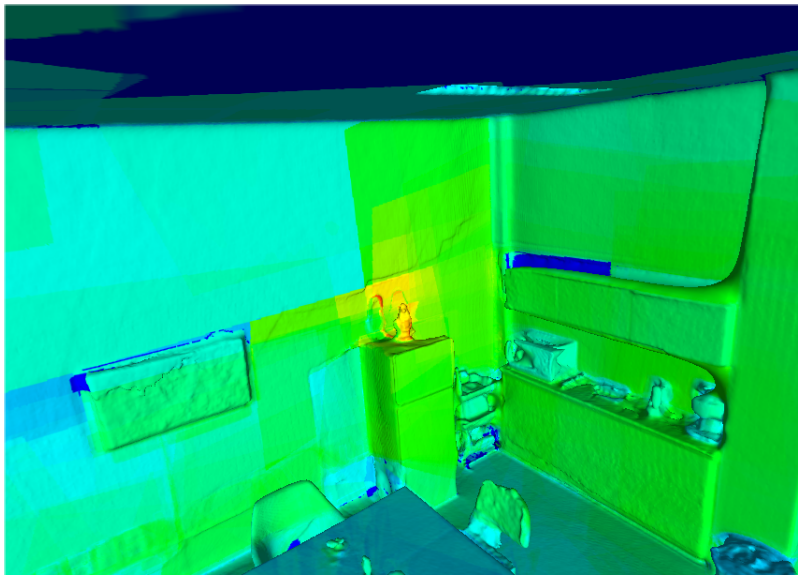


(b) Query 9: CLIP and DinoV2 Fusion Result

Figure 23: Results for Query 9: “Where can I put the trash?”



(a) Query 10: CLIP Result



(b) Query 10: CLIP and DinoV2 Fusion Result

Figure 24: Results for Query 10: “Where is the gnome?”

CLIP and DinoV2 Fusion Result (Figure 24b): In the CLIP-DinoV2 fusion result, the selection is notably more precise. Although no part of the gnome is marked in red (indicating the highest relevance), the object is still correctly identified without extensive irrelevant surroundings.

In this case, both models achieve an acceptable selection of the gnome, demonstrating that the query yields a satisfactory result even with an out-of-place object.

5.3.11 Is there something strange in the room?

The eleventh query, "Is there something strange in the room?", is intentionally ambiguous, with unclear wording and an undefined selection objective. This atypical request is designed to observe how the model responds to a vague question. In this case, there is no specific expected outcome from the model. For Query 11, the results are as follows:

CLIP Result (Figure 25a): Due to the unusual nature of the query, the CLIP selection appears erratic and lacks a coherent focus.

CLIP and DinoV2 Fusion Result (Figure 25b): While the CLIP result seems random, the CLIP-DinoV2 fusion method results in no selection at all, potentially reflecting a more restrained response to the ambiguous query.

In this case, neither method provides meaningful results. The CLIP-DinoV2 fusion may appear slightly less arbitrary due to the absence of any selection, yet the query ultimately yields inconclusive outcomes with both approaches.

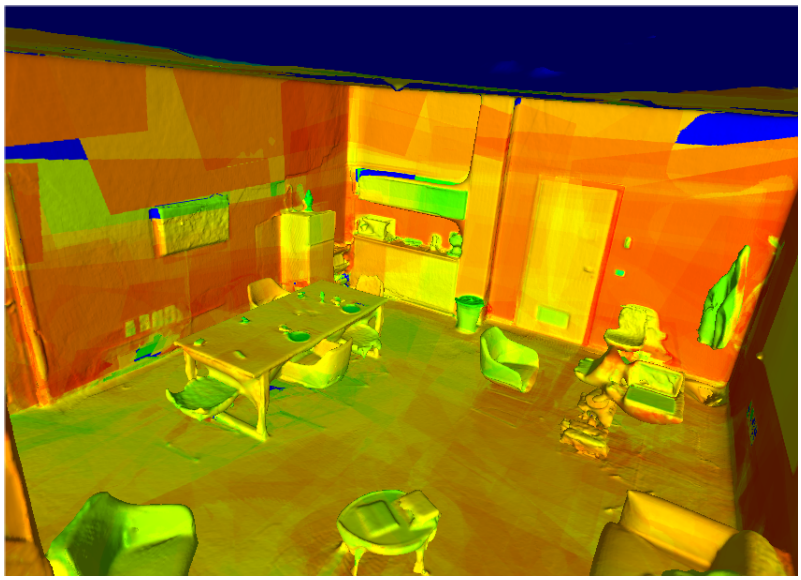
5.3.12 Are there any things out of place?

In the twelfth query, "Are there any things out of place?", we present the model with a similarly vague question to Query 11. The goal of this query is to determine if the model can detect any unusual or outlier elements in the scene without a specific request for particular objects. For Query 12, the results are as follows:

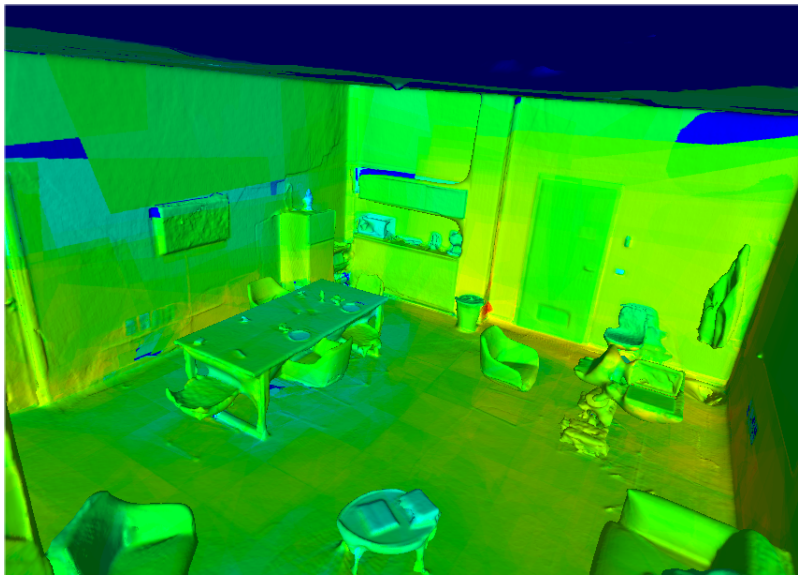
CLIP Result (Figure 26a): As in the previous query (Query 11), the selection appears erratic and random, with no specific objects or areas highlighted.

CLIP and DinoV2 Fusion Result (Figure 26b): Similarly, the CLIP-DinoV2 fusion method yields no significant selection, with no specific elements identified as out of place.

Overall, it appears that the model struggles with handling broad requests to identify objects that may be "out of place," at least within the scope of this dataset. This type of query does not yield meaningful results with either method.

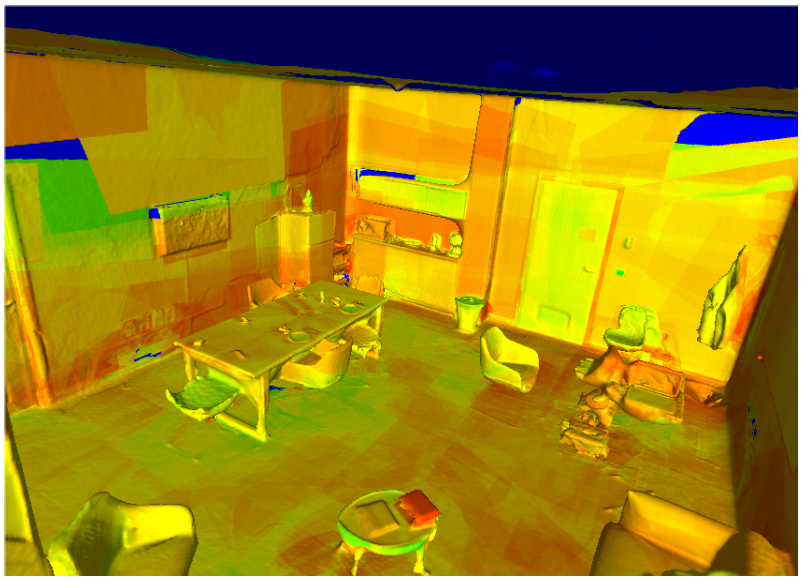


(a) Query 11: CLIP Result

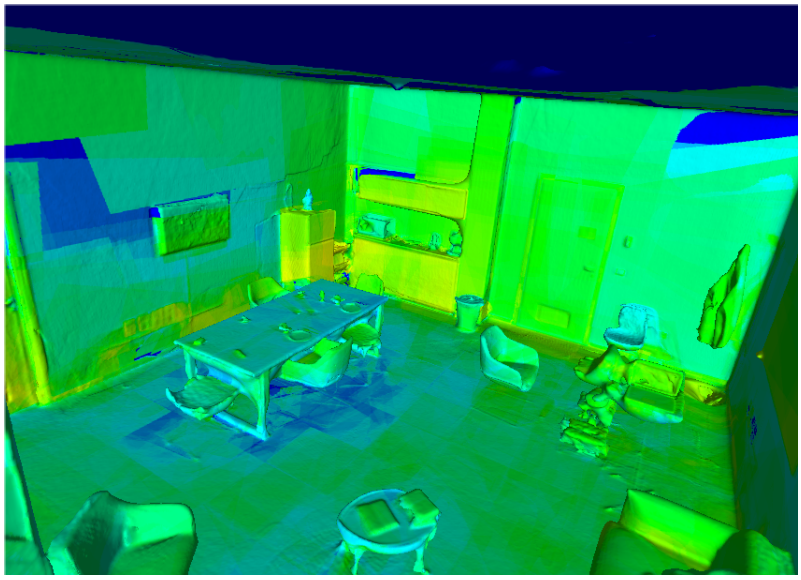


(b) Query 11: CLIP and DinoV2 Fusion Result

Figure 25: Results for Query 11: “Is there something strange in the room?”



(a) Query 12: CLIP Result



(b) Query 12: CLIP and DinoV2 Fusion Result

Figure 26: Results for Query 12: “Are there any things out of place?”

5.4 Limitation and failure analysis

The analysis of the results across the twelve queries reveals several limitations and areas for potential improvement in the CLIP and CLIP-DinoV2 fusion methods. Below is an outline of these limitations, along with failure points and suggested fixes:

Spatial Awareness and Positional Queries One limitation of the model is its difficulty handling spatial qualifiers, as seen in Query 7 ("Where is the left couch?"). Neither CLIP nor CLIP-DinoV2 fusion could accurately distinguish between objects based on spatial information, often selecting all instances of the object rather than discerning based on "left" or "right." This suggests that the model lacks an effective spatial reasoning component, which is crucial for queries involving positional relationships.

Handling of Vague or Ambiguous Queries Queries with broad or ambiguous wording, such as Query 11 ("Is there something wrong in the room?") and Query 12 ("Are there any things out of place?"), yielded erratic or non-specific results, with no meaningful focus on either CLIP or CLIP-DinoV2 fusion. The model appears to struggle with undefined or subjective concepts without a clear reference object or action. This limitation indicates that the current model framework may not be well-suited for processing open-ended or subjective queries.

Precision in Object Segmentation In many cases, the CLIP model alone showed overly broad selections that included surrounding areas or unrelated objects, such as the walls behind objects (seen in Queries 1, 2, and 9). Although CLIP-DinoV2 fusion generally improved precision, it was sometimes inconsistent, as in Query 9, where the wall behind the trashcan was incorrectly selected. This suggests that the fusion method may need refinement to better isolate relevant object boundaries, particularly in complex or cluttered environments.

Detection of Functional Areas The model shows limitations in recognizing functional areas based on object groupings, as evidenced in Query 8 ("Where is the kitchen?"). While it could identify certain objects within the kitchen, it struggled to group them as a distinct area. This suggests a lack of contextual grouping capability, limiting the model's ability to respond to queries that require area-based understanding.

Chapter 6 Conclusion

This thesis project aimed to bind semantic features to complex, reconstructed 3D environments and enable interaction with these environments through natural language queries. The objective was not only to develop such a system but also to ensure it operated in a zero-shot setting, leveraging the pre-trained weights of foundational models. This approach streamlines the utilization of advanced AI models, avoiding unnecessary training time while maintaining high-quality results.

The entire pipeline, transitioning from a scanned environment to the final query result, is detailed in section 4 and proceeds as follows: First, the environment is scanned using the HoloLens 2 device [24], and the acquired scans are utilized to construct a 3D mesh via the Poisson Surface Reconstruction algorithm [12]. Subsequently, a depth map is computed, as described in Algorithm 1. The RGB data captured by the HoloLens is segmented to generate object masks using the approach outlined in [14], which aids in improving the feature extraction process. Features are extracted using either CLIP [18] or DinoV2 [19], as elaborated in Algorithm 2. These features are then projected onto the reconstructed mesh, as described in Algorithm 3.
The final stage involves processing natural language queries, as explained in Algorithm 4, to facilitate querying. This querying phase can utilize either CLIP Algorithm 5 or a hybrid CLIP-DinoV2 approach Algorithm 7. The system then identifies sections of the mesh with the highest similarity scores to the query features and visually highlights the corresponding regions.

In conclusion, the proposed approach represents a significant step toward the development of a system capable of effectively working with real-world environments. The results obtained from the system are consistent and demonstrate its potential. While further refinements and enhancements could improve its overall performance, the capabilities demonstrated by this approach are highly promising and lay a solid foundation for future advancements in this domain.

6.1 Future work

Integration of Spatial Reasoning Mechanisms To improve responses to spatial queries, integrating spatial reasoning layers or positional encoding could enhance the model's ability to interpret directional or positional qualifiers (e.g., "left," "right"). This would enable more precise handling of queries that involve relative positioning within a room.

Refining Object Boundary Detection in Fusion Method Improving the CLIP-DinoV2 fusion method's ability to detect accurate boundaries, particularly in cluttered or complex scenes, could help eliminate unintended selections. Using tighter segmentation algorithms or additional refinement steps post-selection may yield a more precise focus on the intended objects while excluding irrelevant areas.

Training on Functional Area Recognition Enhancing the model's functional area recognition could involve training with labeled zones (e.g., kitchens, offices) defined by sets of objects. This would help the model learn to group objects by purpose rather than individual characteristics alone, enabling it to interpret queries about areas more accurately.

The identified limitations reveal that while the model demonstrates robust object detection capabilities, it requires further refinement in spatial awareness, area-based understanding, and handling of ambiguous queries. The suggested improvements provide a foundation for addressing these limitations and enhancing the model's overall responsiveness and accuracy in complex or undefined query scenarios.

References

[1] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[2] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. Meshlab: an open-source mesh processing tool. volume 1, pages 129–136, 01 2008.

[3] THE VISUAL COMPUTING LAB ISTI CNR. The visualization and computer graphics library, (vcglib in short). More info on: `https://vcg.isti.cnr.it/vcglib/`.

[4] Thompson et al. *Manual of photogrammetry.* ASPRS, 3 edition, 1966.

[5] Paolo Fasano. Embreeadaptor: an adaptor to use intel embree in vcglib. More info on: `https://github.com/cnr-isti-vclab/vcglib/blob/main/wrap/embree/EmbreeAdaptor.h`.

[6] Wolfgang Förstner and Bernhard P. Wrobel. *Photogrammetric computer vision.* Springer, 2016.

[7] Fabio Ganovelli, Massimiliano Corsini, and Sumanta Pattanaik. *Introduction to Computer Graphics: A Practical Learning Approach.* Taylor & Francis Inc, 2014.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[9] Geoffrey Hinton, li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Phuongtrang Nguyen, Tara Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29:82–97, 11 2012.

[10] Yining Hong, Chunru Lin, Yilun Du, Zhenfang Chen, Joshua B. Tenenbaum, and Chuang Gan. 3d concept learning and reasoning from multi-view images, 2023.

[11] Krishna Murthy Jatavallabhula, Alihusein Kuwajerwala, Qiao Gu, Mohd Omama, Tao Chen, Shuang Li, Ganesh Iyer, Soroush Saryazdi, Nikhil Keetha, Ayush Tewari, Joshua B. Tenenbaum, Celso Miguel de Melo, Madhava Krishna, Liam Paull, Florian Shkurti, and Antonio Torralba. Conceptfusion: Open-set multimodal 3d mapping. *Robotics: Science and Systems (RSS)*, 2023.

[12] Michael M. Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In Alla Sheffer and Konrad Polthier, editors, *Proceedings of the*

Fourth Eurographics Symposium on Geometry Processing, volume 256 of *SGP '06*, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[13] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded radiance fields, 2023.

[14] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv:2304.02643*, 2023.

[15] Jatavallabhula Krishna Murthy, Soroush Saryazdi, Ganesh Iyer, and Liam Paull. gradslam: Dense slam meets automatic differentiation. In *arXiv*, 2020.

[16] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning, 2015.

[17] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., USA, 1 edition, 1997.

[18] OpenAI. Clip - contrastive language-image pre-training. More info on: `https://github.com/openai/CLIP`.

[19] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2024.

[20] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[21] 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. *CoRR*, abs/2103.00020, 2021.

[22] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[23] Omurhan A. Soysal and Mehmet Serdar Guzel. An introduction to zero-shot learning: An essential review. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–4, 2020.

[24] Dorin Ungureanu, Federica Bogo, Silvano Galliani, Pooja Sama, Xin Duan, Casey Meekhof, Jan St{"uhmer, Thomas J. Cashman, Bugra Tekin, Johannes L. Sch{"onberger, Bugra Tekin, Pawel Olszta, and Marc Pollefeys. HoloLens 2 Research Mode as a Tool for Computer Vision Research. *arXiv:2008.11239*, 2020.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[26] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph.*, 33(4):143:1–143:8, July 2014.

[27] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine*, 13:55–75, 08 2018.

Chapter 7 Appendix A: Extended table of contents

List of Figures

List of Tables