A. Limitations

In this section, we discuss the limitations of our approach. First, there is no formal guarantee that the random view sampling algorithm covers each triangle. We mainly used this choice to be compatible with previous work. Alternatively, one could devise a greedy view selection algorithm that ensures each triangle is selected once. Another approach would be to use a set of uniformly distributed viewpoints that are fixed for the complete dataset. Second, we are not able to test other types of large language models because the latest versions are not publicly available [99]. Third, GLIP prediction is not perfect. For example, it can incorrectly detect parts that are semantically different but still look similar (for instance, the front and the backpack of an astronaut, see Figure 1).

B. SATR Pseudocode

In Algorithm 1, we show the pseudocode of our proposed method (SATR).

Algorithm 1: Segmentation Assignment with Topological Reweighting (SATR) in highlevel pseudocode.

```
Input : Zero-shot 2D object detector $D(\boldsymbol{x}, \boldsymbol{t})$,
        where $x$ is an image and $t$ is a text prompt.
Input : Shape surface as a mesh of faces $\mathcal{F}$.
Input : Set of textual prompts $\mathcal{T}$ representing
        semantic regions/classes.
Input : Number of views $N_{views}$
Input : The q-ring neighborhood $q$.
Output: The predicted semantic label of each face
        of the input mesh $\mathcal{F}$.
# Initialize face scores
$S$ = zeros($\mathcal{F}$.faces.length, $\mathcal{T}$.length),
# Compute the pair-wise geodesic distance between
  every pair of faces
$G$ = computeFacePairwiseDistance($\mathcal{F}$)
# Find the $q$-ring neighborhood for all the faces of
  the mesh
$Q$ = getFaceQNeighbors($\mathcal{F}$, q)
# Render the mesh
$V$, Pixel2Face = renderMesh($\mathcal{F}$, $N_{views}$)
for $v$ in $V$ do
    for $t$ in $\mathcal{T}$ do
        # Detect 2D Bounding Boxes for the given $t$
          prompt
        $B_{v,t}$ = predictBoxes($v$, $t$)
        for $b_{t,v}$ in $B_{v,t}$ do
            # Get the visible faces inside $b_{t,v}$
            $f_{t,v}$ = getVisibleFaces($b_{t,v}$, $\mathcal{F}$,
              Pixel2Face)
            # Compute capital face
            $c$ = computeCapitalFace($f_{t,v}$, $\mathcal{F}$)
            # Compute frequency of the visible faces
            $w_{freq}$ = faceFreq($f_{t,v}$, Pixel2Face)
            # Compute Gaussian Geodesic weights
            $w_{geo}$ = faceGeodesicWeights($c$, $f_{t,v}$)
            # Compute visibility smoothing weights
            $w_{vis}$ = faceVisibilityWeights($f_{t,v}$, $Q$)
            # Update face scores
            $S_{f_{t,v}, i_t}$ += $w_{geo}$ * $w_{vis}$ * $w_{freq}$
        end
    end
end
face_label = argMax($S$, axis=1)
Return face_label;
```

C. Additional Ablation Studies

C.1. Changing the Number of Views

We investigate the effect of changing the number of input rendered views on our proposed method SATR. In Table 3, we report the mIoU performance on both the coarse and the fine-grained FAUST benchmarks. Generally, increasing the number of views results in better segmentation performance in both benchmarks. However, the computation time increases, and there are diminishing returns for adding a large number of views.

C.2. Camera View-port Sampling Approach

We ablate using different sampling approaches for choosing the camera view-ports. In Table 1, we compare between sampling camera view-ports using the approach described in [22] and doing uniform viewpoint sampling using a range of equidistant elevation and azimuth angles. We report the mIoU performance on both the coarse and the fine-grained FAUST benchmarks. We observe that sampling from a normal distribution ($\mu = 3.14$, $\sigma = 4$) results in better performance in both benchmarks. The reason behind this is having more control over the range of the elevation and azimuth angles can produce better views that cover most of the input mesh and avoid sampling views where a lot of occlusions may happen. More complex view selection could be future work.

C.3. Coloring The Input Mesh

We ablate the effect of changing the color of the input mesh on the performance of SATR. We run three experiments by using four different colors for the input meshes; grey, red, blue, and natural skin color. As shown in Table 2, we find that using the gray color results in the best performance compared to other colors.

	Coarse-Grained mIoU	Fine-Grained mIoU
Uniform Sampling	81.58	43.76
Sampling using Normal Distribution [23]	**82.46**	**46.01**

Table 1: Ablation on using different view-port sampling methods in our proposed method SATR on FAUST coarse and fine-grained benchmarks. (with ten rendered views as input).

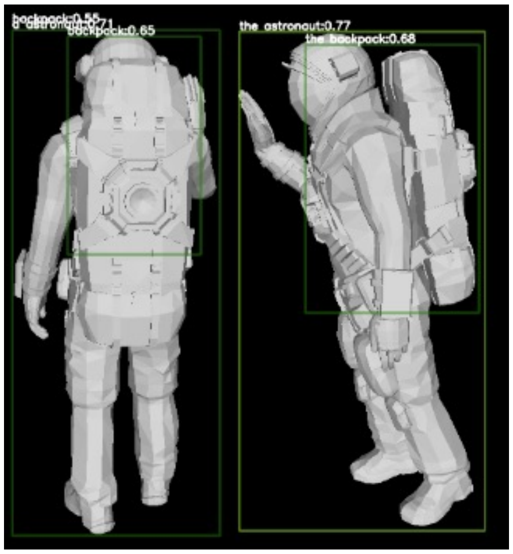

Figure 1: GLIP model can incorrectly detect semantically different parts that look similar. Since the parts of the astronaut suit look similar, GLIP mistakenly predicts the astronaut's chest as a part of the backpack and doesn't predict a tighter bounding box around the backpack.

	Coarse-Grained mIoU	Fine-Grained mIoU
Red	80.62	40.67
Blue	81.35	42.58
Skin color	82.22	44.58
Gray	**82.46**	**46.01**

Table 2: Ablation on changing the vertex colors of the input 3D models to our proposed method SATR on FAUST coarse and fine-grained benchmarks. (with ten rendered views as input).

C.4. Summation of Reweighting Factors

We replace the multiplication of both of the Gaussian Geodesic and Visiblity Smoothing reweighting factors as shown in Equation (12) with addition:

$$\mathcal{W}^*_m[n,k] = \sum_{\ell=1}^{L_{m,k}} \mathcal{W}^\ell_m[n,k,\ell] \times s^n_{m,\ell} \times (r^{m,k}_\ell + \boldsymbol{v}^{m,k}[n]). \tag{13}$$

# Views	Coarse-Grained mIoU	Fine-Grained mIoU
5	53.99	25.96
10	82.46	24.3
15	80.48	43.40
20	82.41	45.20
30	84.06	**47.87**
40	**84.26**	47.67

Table 3: Ablation on using a different number of rendered views as input to our proposed method SATR on FAUST coarse and fine-grained benchmarks.

	Backbone	Coarse-mIoU	Fine-mIoU
SATR-F	GLIP [55]	81.16	41.96
SATR (add)	GLIP [55]	**82.75**	44.90
SATR (mul)	GLIP [55]	82.46	**46.01**

Table 4: Ablation on different ways of combining the reweighting factors.

We show the mIoU performance on both FAUST coarse and fine-grained benchmarks in Table 4. The addition of the reweighting factors gave slight increase in performance for the coarse benchmark while perform significantly worse than multiplying the reweighting factors.