MVA-NPM3D Project Final Report

Paper review : Efficient 3D Semantic Segmentation with Superpoint Transformer

Martin Jolif & Mike Huttenschmitt

March 20, 2025

1 Introduction and purpose

We review and experiment with the technology introduced in "Efficient 3D Semantic Segmentation with Superpoint Transformer" [4] for point cloud semantic segmentation. Instead of classifying individual points, the approach classifies clusters of points (called superpoints), which are grouped based on shared features (geometry, radiometry). This clever representation significantly reduces the number of clusters compared to individual points, leading to faster computations. By using fewer superpoints—which are small, meaningful groups of points—the model is able to leverage transformer architectures while remaining lightweight. Each cluster is classified first, and then the label is propagated to all points within the cluster. While this method might seem restrictive, it proves effective in practice. The approach achieves comparable results to the Stratified Transformer [1] on the S3DIS dataset in terms of mean Intersection over Union (mIoU), but with 20 times faster training and over 40 times fewer model parameters, demonstrating its efficiency and scalability.

Model and pipeline

The goal of the Superpoint Transformer (SPT) model approach presented in this paper is first to create a partition from a 3D point cloud with pieces of locally homogeneous geometry and radiometry called superpoints. If these superpoints are well constructed, since they should not break the object boundaries, we just have to give one class to the superpoint and implicitly all the points in the superpoint will have the same class. This first part is done during the pre-processing. Secondly, we need to learn to classify the superpoints. At the end we have done point cloud semantic segmentation.

2 Preprocessing

2.1 Tiling & voxelization

First, it consists of applying **tiling** using `SampleXYTiling` for too large point clouds, for example, big outdoor datasets. Then, we apply **voxelization** using `gridsampling3d` to

regulate point density based on the scenario, typically 1 cm for indoor environments and 10–25 cm for outdoor scenes. This gives $\boldsymbol{\mathcal{P}_0}$, the first level of partition.

2.2 First level of the hierarchical partitioning

We initialize with the trivial partition $\mathcal{P}_0 = \{\{x\} \mid x \in \mathcal{X}\}$ described above, illustrated in figure 5, where each (super)point $c \in \mathcal{C}$ has a feature vector f_c of dimension D. The feature vectors f_c include radiometric features (RGB colors or intensity) and geometric features (PCA-based features like linearity, planarity, scattering, verticality) computed with KNN.

The corresponding adjacency graph $\mathcal{G} = (\mathcal{C}, \mathcal{E}, w)$ is created using the k-nearest neighbor graph ($k = 10$), with uniform edge weights $w_{u,v} = 1$.

2.3 Hierarchical superpoint partitioning

Our objective is now to obtain a **hierarchical superpoint partitioning** of the point cloud like in figure 6:

1. $\boldsymbol{\mathcal{P}_0}$ – The voxelized point cloud.
2. $\boldsymbol{\mathcal{P}_1}$ – Initial superpoint clustering.
3. $\boldsymbol{\mathcal{P}_2}$ – A coarser clustering of superpoints.

Under a mathematical point of view a hierarchical partitioning is defined as follows:

Definition 1 (Hierarchical Partitions)**.** *A sequence of partitions $\mathcal{P} := [\mathcal{P}_0, \ldots, \mathcal{P}_I]$ is said to be a hierarchical partition of $\mathcal{X}$ if $\mathcal{P}_0 = \{\{x\} \mid x \in \mathcal{X}\}$, and for each $i \in [0, I-1]$, $\mathcal{P}_i$ is a partition of $\mathcal{P}_{i+1}$, meaning that each subset in $\mathcal{P}_{i+1}$ is formed by merging subsets of $\mathcal{P}_i$.*

As it reverses the hierarchy in the paper definition, we modify it in order to align with the illustrations and ensure coherence with other sections. This modification ensures a consistent hierarchy where finer details are preserved at lower levels before being aggregated.

This step is performed with the `cutpursuitpartition` algorithm. It consists of alternately performing **graph construction** and **superpoint aggregation** in order to compute the three levels of partitioning. This iterative process progressively merges points into superpoints while preserving geometric structures.

2.4 Energy-Based Partitioning

To construct a coarser partition, we approximate the feature function f_c by searching for a piecewise-constant function on the graph. All is done by minimizing the following energy:

$$J(e; f, \mathcal{G}, \lambda) = \|e - f\|^2 + \lambda \sum_{(u,v)\in\mathcal{E}} w_{u,v} \cdot [e_u \neq e_v], \tag{1}$$

where $e \in \mathbb{R}^{D\times|\mathcal{C}|}$ is the approximated piecewise-constant signal, $f \in \mathbb{R}^{D\times|\mathcal{C}|}$ is the original feature signal, $\mathcal{G} = (\mathcal{C}, \mathcal{E}, w)$ is the adjacency graph, λ controls the balance between fidelity and smoothness, and $[e_u \neq e_v]$ is the indicator function which penalizes discontinuities in e.

Minimizing this energy yields a new coarser partition $\mathcal{P}_e$, where e is constant within each superpoint of $\mathcal{P}_e$. The new feature representation is given by the values of e within each superpoint.

2.5 Superpoint Graph Construction

From the optimized partition $\mathcal{P}_e$, we construct a new adjacency graph $\hat{\mathcal{G}}_e = (\mathcal{P}_e, \mathcal{E}_e, w^e)$ for the superpoints, where:

$$\mathcal{E}_e := \{(U, V) \mid U, V \in \mathcal{P}_e, (U \times V) \cap \mathcal{E} \neq \emptyset\}. \tag{2}$$

The weight of each edge is defined as:

$$w^e_{U,V} := \sum_{(u,v) \in U \times V \cap \mathcal{E}} w_{u,v}. \tag{3}$$

This hierarchical process enables to obtain a new representation of the previous partition, reducing the amount of superpoints, and connect them through a graph. Now, we have $\mathcal{P}_1$ and we can apply this process another time to obtain the next superpoint partition.

3 The Superpoint Transformer Model

The deep model architecture benefits from this hierarchical structure, significantly reducing its size. Similar to a **U-Net**, it follows a downsampling and upsampling path:

- **Encoding**: $\mathcal{P}_0 \rightarrow \mathcal{P}_1 \rightarrow \mathcal{P}_2$
- **Decoding**: $\mathcal{P}_2 \rightarrow \mathcal{P}_1$

Unlike U-Net, which downsamples through grid-based subsampling, this approach **leverages real geometric structures** rather than abstract coordinate-based partitioning. There's no need to upsample back to $\mathcal{P}_0$, as all points within a superpoint in $\mathcal{P}_1$ inherit the same label.

The pipeline processes the point cloud with features, followed by an MLP (similar to PointNet [3]) to extract additional features. Then, it applies pooling and a transformer to obtain $\mathcal{P}_1$, followed by another pooling and attention step to produce $\mathcal{P}_2$. The decoding phase involves unpooling, combined with skip connections from $\mathcal{P}_1$, to reconstruct the final output, as illustrated in Figure 1.

To predict the class of the superpoints, the model performs **graph node classification** using a graph-based transformer, which is a variant of the **Graph Attention Networks (GAT)** [7].

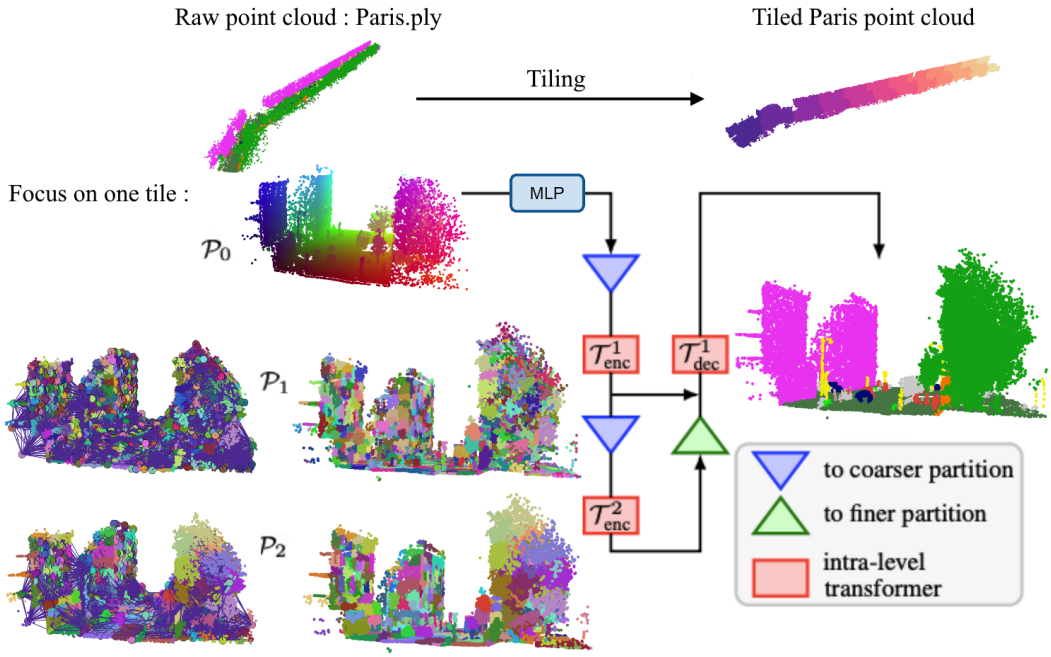


Figure 1: Training pipeline

3.1 Encoder

At each stage i, the encoder computes features for superpoints. For a superpoint p, its feature is the relative position of p to its parent concatenated with the maximum feature of its children. These features are passed through an MLP ϕ^i_{enc} $\forall i \geq 0$ and further refined by a transformer $\mathcal{T}^i_{enc}$ $\forall i \geq 1$ to gather information from neighbors.

For each stage i, the updated features g^p_i for each superpoint $p \in \mathcal{P}_i$ are:

$$g^i_p = \mathcal{T}^i_{enc} \circ \phi^i_{enc} \left([x^i_p, \max_{q \in \text{children}(p)} g^{i-1}_q] \right)$$

Where ϕ^i_{enc} is an MLP function, $\mathcal{T}^i_{enc}$ is the transformer module, x^i_p is the relative position of superpoint p at stage i, g^{i-1}_q is the feature of superpoint q from the previous stage.

3.2 Decoder

The decoder updates features by using information from both the current and next stages. For each superpoint p, the updated feature h^i_p is computed as:

$$h^i_p = \mathcal{T}^i_{dec} \circ \phi^i_{dec} \left([x^i_p, g^i_p, h^{i+1}_{\text{parent}(p)}] \right)$$

Where ϕ^i_{dec} is an MLP function, $\mathcal{T}^i_{dec}$ is an attention-based module, x^i_p is the relative position of superpoint p at stage i, g^i_p is the feature of superpoint p at stage i from the encoder, $h^{i+1}_{\text{parent}(p)}$ is the feature of the parent superpoint in the next stage.

3.3 Transformer module

The transformer module $\mathcal{T}$ uses self-attention to propagate information between neighboring superpoints at each stage of the encoder and decoder. The proposed attention scheme is similar to the classic one (GAT) [7] with only two differences. The queries adapt to each neighbor and the softmax normalization is done with the neighborhood size instead of the key dimension.

4 Experiments with provided datasets

4.1 S3DIS and DALES datasets

From an experimental perspective, we relied on the code provided by the authors in their GitHub repository as mentioned in their paper. The configuration required to run the model demands 64GB of RAM and a GPU. Therefore, we decided to use an AWS virtual machine with the necessary configuration to perform both model training and inference.

We then trained the SPT model on the 5th fold of the S3DIS dataset as well as on the DALES dataset and obtained the following results (see Table 1 and Table 2).

	Beam	Board	Bookcase	Ceiling	Chair	Clutter	Column	Door	Floor	Sofa	Table	Wall	Window	**Mean**
Our Training	0.2	65.0	72.6	92.9	89.4	57.7	44.1	66.8	97.6	80.1	77.9	83.2	56.8	68.0
Paper	0.2	63.1	73.2	92.6	88.8	60.0	42.0	67.1	97.7	86.0	81.0	83.5	60.6	68.9

Table 1: Results for SPT on the 5th fold of S3DIS dataset

	Building	Car	Fence	Ground	Pole	Power line	Truck	Vegetation	**Mean**
Our Training	96.9	86.2	50.5	96.8	64.0	93.8	48.3	93.1	78.7
Paper	96.7	86.1	52.7	96.7	65.3	94.0	52.4	93.1	79.6

Table 2: Results for SPT on DALES dataset

For each dataset, the results in both columns are closely aligned, indicating that we have successfully replicated the findings from the paper. The training process was relatively short compared to other state of the art methods taking only a few hours on AWS ($\approx 3h$ for 5th fold of S3DIS and $\approx 8h$ for DALES). This confirms the low training time stated in the paper.

4.2 Vancouver dataset

Then, we decided to test the previously trained model on the DALES dataset, on another similar outdoor dataset. We choose the Vancouver dataset which contains LiDAR data of the City of Vancouver covering a total of 134 square kilometers. We arbitrarily choose the 1km^2 tile containing the Thunderbird Stadium, the Brockhouse Park and the University Hill Secondary School to ensure having several types of props.

Sub-tile location Ground truth Prediction

Figure 2: Prediction on a sub-tile of Vancouver $1km^2$ tile

We used the checkpoint from the SPT model trained on the DALES dataset and made a correspondence between the classes of the DALES and Vancouver datasets. This experiment allowed us to see if SPT can generalize. To avoid memory issues, we have done the inference only on a tile from the original $1km^2$ square (red square on the left point cloud). We can observe that qualitatively the prediction seems good. It can distinguish the ground, the vegetation and buildings. We get that **70%** of the voxels in that tile point cloud are classified correctly (at the $\mathcal{P}_0$ level). The code to obtain these results is in this notebook.

5 Our experiment on Paris-Lille-3D

5.1 The dataset

We uses the 10-classes version of the Paris-Lille-3D dataset [6], which is composed of four .ply files: Paris, Lille2, Lille1_1 and Lille1_2. This files can be opened in Cloud Compare, the scalar field called "class" represent the label of each point in the point cloud, here is a visualization of Lille1_1 (see figure 3).

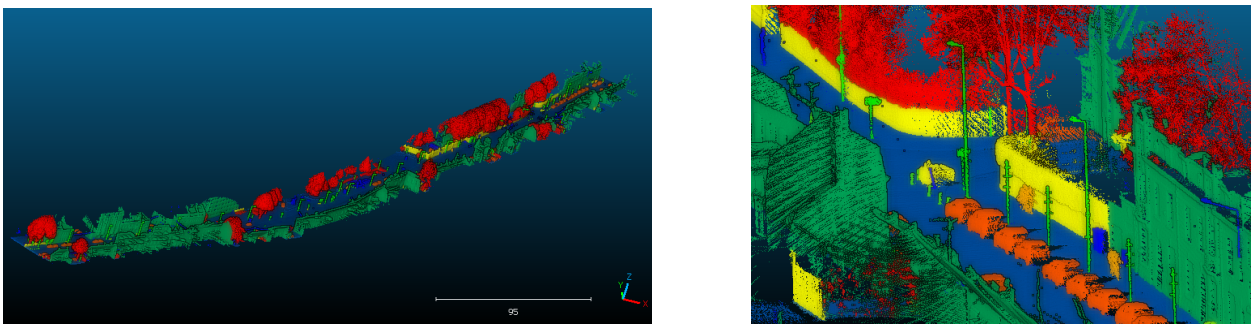


Figure 3: Data from Lille1_1.ply file

The 10 classes are: unclassified, ground, building, pole/road sign/traffic light, bollard/small pole, trash can, barrier, pedestrian, car and natural/vegetation. We can observe that green corresponds to buildings, light orange to pedestrians, orange to cars etc.

5.2 Created files in the project

To be able to train and test the model on the Paris-Lille-3D dataset we needed to create/modify some python files from the original repository. All the exact creations/modifications, can be seen in the last commits of our fork of the repository. We also listed all the python files in the Appendix in section A.1. Our code for the Paris-Lille-3D dataset is highly inspired by the code for the DALES dataset.

5.3 Full training of SPT on Paris-Lille-3D

We split our dataset as follows: Paris.ply and Lille1_1.ply for the training set, Lille2.ply for the validation set, and Lille1_2.ply for the test set, because the script requires a valid and a test set both labeled.

5.3.1 First training

For the first training we ran the training python file by keeping the parameters of training and preprocessing the same as for the DALES dataset.

Here are some visualizations to understand how the trained model performs. We first, apply some tiling along the xy-axis:

Figure 4: Raw data point cloud and tiled version

Here is a plot with the ground truth displayed. This tile is at coordinates (x,y) = (0,1). It's a good tile example because we can see a wide variety of props, roads and vegetation.

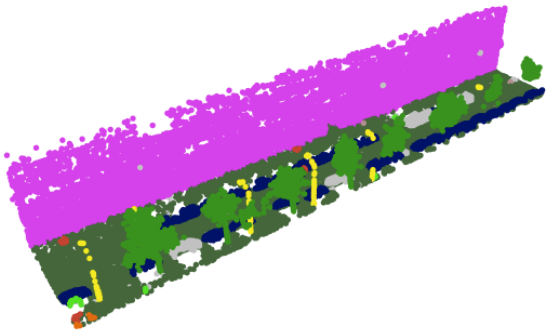
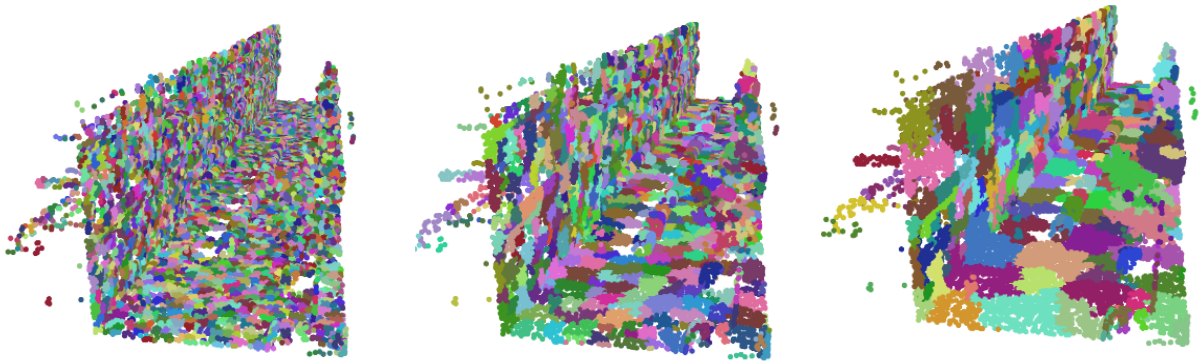


Figure 5: One tile of the original point cloud with ground truth labels

Some of the parameters chosen for the pre-processing are the following:

- voxelization: 10cm cubes.
- local geometric point features: reflectance, linearity, planarity, scattering, verticality, elevation
- adjacency graph constructed with KNN-based neighbor search (K = 25)

After, the pre-processing steps, we obtain, the following partition levels associated with superpoint adjacency graphs that will be processed by the SPT model.



Partition Level 0 ($\mathcal{P}_0$) Partition Level 1 ($\mathcal{P}_1$) Partition Level 3 ($\mathcal{P}_2$)

Figure 6: Different levels of NAG, with coarser superpoints on the right.

We can also have a look at the predictions and the 3D features, and it is incredible how we can already differentiate shape just with the color mapping

Figure 7: Predictions and 3D features

It seems we made an error during our definition of the Paris-Lille dataset in the parisLille.yaml and the ParisLille_config.py files. Indeed, as we can see, the predicted labels never predict vegetation as it should be the case. The mean IoU achieved is of 52%, the complete result can be seen in Table 3.

5.3.2 Second training

This time, we correct the number of labels in the SPT model such that it can predict vegetation. We also change the tiling to only tile along the y-axis instead of the xy-axis and keep the same pre-processing parameters as before. The mean IoU achieved is of 55%, the complete result can be seen in Table 4.

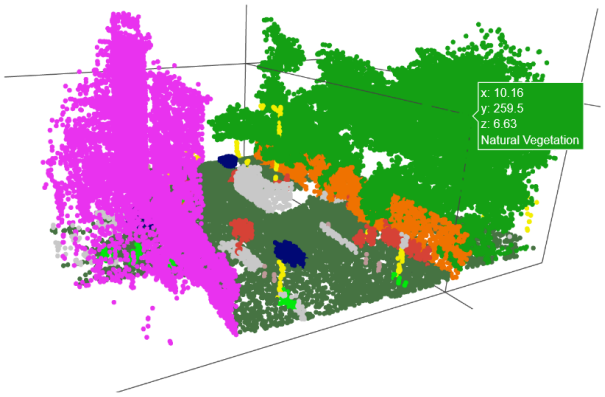


Figure 8: Vegetation well predicted by the model

5.3.3 Third training

When correcting the label predictor in the second training we gain 3%, however we expected to get a better score with this correction. So we tried to tweak the hyper-parameters of the training, knowing that DALES is an aerial scan and that KITTI is a car-mounted scan, we adjusted the configuration file to be closer to KITTI's. Unfortunately parameters from KITTI creates heavier preprocessed data leading to memory issues in the training loop.

6 Limitations, Advantages and Enhancement

- Voxelisation and superpoints both impact oracle accuracy (oracle accuracy corresponds to the highest achievable performance with the partition at hand). The authors report 85% oracle accuracy on S3DIS when assigning the most frequent voxel label to each point of the voxel. Superpoints also reduce inconsistencies but preserves semantics, while voxelization arbitrarily cuts along axes. With 15 cm superpoint clusters, oracle accuracy can reaches 98%.

- Clustering limits the performance, as seen in the maximum score achievable by oracle. So an idea would be to compute an agreement score based on the entropy of label prediction for superpoints, and use this score to predict different classes inside the same superpoint [5].

- Point features are important, indeed, without handcrafted features, the SPT model performance is decreased by : -0.7 mIoU on S3DIS, -4.1 mIoU on KITTI and -1.4 mIoU on DALES.

- Edges on graph have huge influence, indeed the authors announce that removing the adjacency encoding between superpoints leads to a significant drop of 6.3 points on S3DIS.
- An idea to enhance the model could be to use a lightweight deep learning model that learns local features to enhance semantic information instead of handcrafted features as in [2].

7 Conclusion

The SPT method presented is lightweight and performs well on DALES and S3DIS, requiring only 5 hours for full re-training in our experiments. We successfully adapted and trained it on Paris-Lille, despite significant differences in dataset structure—fewer points overall and subtler class distinctions (e.g., poles, trash cans, pedestrians). The model struggled most with these fine-grained labels with 14% IoU scores but achieved around 75% IoU for buildings, ground, and cars, this leads to a final score mIoU of 55%, while it is lower than other methods on Paris-Lille score-board we deem this score can be enhanced with more data and with parameter tuning. The key takeaway is the efficiency and competitiveness of SPT. It matches or surpasses other models while being significantly lighter.

References

[1] Xin Lai, Jianhui Liu, Li Jiang, Liwei Wang, Hengshuang Zhao, Shu Liu, Xiaojuan Qi, and Jiaya Jia. Stratified transformer for 3d point cloud segmentation, 2022.

[2] Loic Landrieu and Mohamed Boussaha. Point cloud oversegmentation with graph-structured deep metric learning, 2019.

[3] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017.

[4] Damien Robert, Hugo Raguet, and Loic Landrieu. Efficient 3d semantic segmentation with superpoint transformer, 2023.

[5] Damien Robert, Hugo Raguet, and Loic Landrieu. Scalable 3d panoptic segmentation as superpoint graph clustering, 2024.

[6] Xavier Roynard, Jean-Emmanuel Deschaud, and François Goulette. Paris-lille-3d: a large and high-quality ground truth urban point cloud dataset for automatic segmentation and classification, 2018.

[7] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

A. Appendix

A.1 Our code contribution

- **src/datasets/ParisLille.py**: Read/load data from the .ply files, specify scalar fields.
- **src/datasets/ParisLille_config.py**: Dataset spliting into train/val/test, class names and corresponding colors for visualizations.
- **configs/experiment/semantic/parisLille_11g.yaml**: Training parameters, tiling resolution, epochs, model, learning rate.
- **configs/datamodule/semantic/parisLille.yaml**: Preprocessing parameters, scalar fields, computed features, voxelization.
- **src/datamodules/parisLille.py**: Defines a PyTorch Lightning DataModule for the Paris-Lille-3D dataset, which handles data loading/splitting and pre-processing.
- **src/utils/keys.py**: Add reflectance to scalar fields list.
- **configs/datamodule/semantic/_features.yaml**: Add reflectance to scalar field dictionary.
- **src/transforms/sampling.py**: Add a new class SampleYTiling to tile along the y-axis only.
- **src/datasets/base.py**: Enable the use of the SampleYTiling class during the pre-processing.
- **notebooks/Inference_on_vancouver_from_dales_checkpoints.ipynb** and **notebooks/paris-lille.ipynb**: notebooks to obtain visualizations that are used in this report.

A.2 Detailed results on Paris-Lille-3D

	Barrier	Bollard/Small Pole	Building	Car	Ground	Pedestrian	Poal/Road sign/Traffic light	Trash Can	Unclassified	**Mean**
Our Training	35.5	50.8	95.0	74.8	97.6	5.5	79.9	18.4	14.4	52.5

Table 3: Results for SPT on Paris-Lille-3D dataset (1st training)

	Barrier	Bollard/Small Pole	Building	Car	Ground	Vegetation	Pedestrian	Poal/Road sign/Traffic light	Trash Can	Unclassified	**Mean**
Our Training	31.7	52.9	94.1	76.8	97.5	75.8	5.6	74.1	22.6	19.0	55.0

Table 4: Results for SPT on Paris-Lille-3D dataset (2nd training)