

# Supplementary Material

## A EXPERIMENTAL MOTIVATIONS FOR STYLE EMBEDDING

### A.1 TOY EXPERIMENTS ON STATISTICS OF ENCODER FEATURES

We hypothesize that if the strength of sensor-induced structural perturbations can be recovered directly from encoder feature statistics, then these statistics constitute an explicit domain cue for the synthetic-to-real gap. This is because the feature statistics can be considered to effectively contain information about the elements that constitute the synthetic-to-real domain gap. Concretely, following our scene-wise style embedding (SSE), we divide points into radial distance bins and, for each encoder block, compute per-bin channel-wise means and standard deviations; the resulting vectors are concatenated across the five encoder blocks. This is identical to the SSE defined in our method in main paper.

**Setup.** From the *SynLiDAR* dataset, we randomly sample 1,000 scenes and synthetically inject four perturbations known to drive the real-sensor gap: motion blur, rolling shutter, point jitter, and point drop. Each corrupted sample is governed by a scalar factor  $\alpha \in 0.0, 0.25, 0.5, 0.75, 1.0$ . How the perturbation was applied is described in Listing. S1. Features are extracted with a MinkUNet (Choy et al., 2019), using all five encoder blocks. To mitigate depth-dependent sparsity, we adopt radial binning (10,m bins up to 80,m) and form SSE variants (mean, standard deviation) by flattening and concatenation.

**Prediction and evaluation.** We fit a linear ridge regressor to predict  $\alpha$  from the SSE features and report *GroupKFold*  $R^2$ , where folds are defined by *scene IDs* so that all augmentations from the same scene reside in the same fold—preventing leakage across train/test. As a sanity check, we include a *count-only* baseline that uses only a radial histogram of occupied voxels (no features).

**Findings.** Across all four perturbations, SSE features are highly predictive of  $\alpha$  as in scatter plots in Fig. S1). both mean-only and std-only variants typically yield  $R^2 \approx 1.0$ , showing that the relationship between encoder statistics and perturbation strength generalizes to unseen scenes. In contrast, the count-only baseline behaves as expected: it fails when geometric distortions largely preserve point density (e.g., rolling shutter;  $R^2 \approx 0$ ), is only moderate for motion blur, and succeeds only when density is intentionally altered (point drop) or can inflate the number of occupied voxels (strong jitter).

**Takeaway.** Encoder feature statistics (mean and standard deviation aggregated as SSE) carry a direct, quantitative signature of point-structure discrepancy. Reading out  $\alpha$  from unlabeled scenes with near-perfect GroupKFold accuracy supports using SSE as a principled domain cue for adaptation.

### A.2 CLASS-WISE STYLE EMBEDDING

**Observation.** The class-wise *Pearson correlation* matrix of style embeddings (Fig. S2 (a)) shows strong diagonal dominance with clear block structures aligned with semantic superclasses (e.g., road - parking - sidewalk, car - truck - other.vehicle, vegetation - trunk - terrain). While several class pairs are mutually close, pairs with markedly different dynamicity or point density (e.g., car - person - vegetation) exhibit weak correlation, indicating heterogeneous structure statistics across classes. These patterns indicate a class-dependent synthetic-to-real shift. Consequently, a single global alignment is unlikely to correct all classes simultaneously. Using *CSE* better reflects the varying domain gaps for each class, leading to more accurate adaptation.

**Evidence from hierarchical grouping.** The cosine distance distributions (Fig. S2 (b)) exhibit a clear left shift and tighter concentration for intra-superclass pairs relative to inter-superclass pairs, indicating that classes sharing dynamics/point-structure lie closer in the style space. Consistently, the left panel (Fig. S2 (a)) shows a pronounced block-diagonal (“blocky”) correlation pattern aligned with our semantic superclasses—high within-superclass correlation and suppressed cross-superclass correlation. The experimental results in Sec. A.1 indicate that the encoder feature statistics, i.e.,

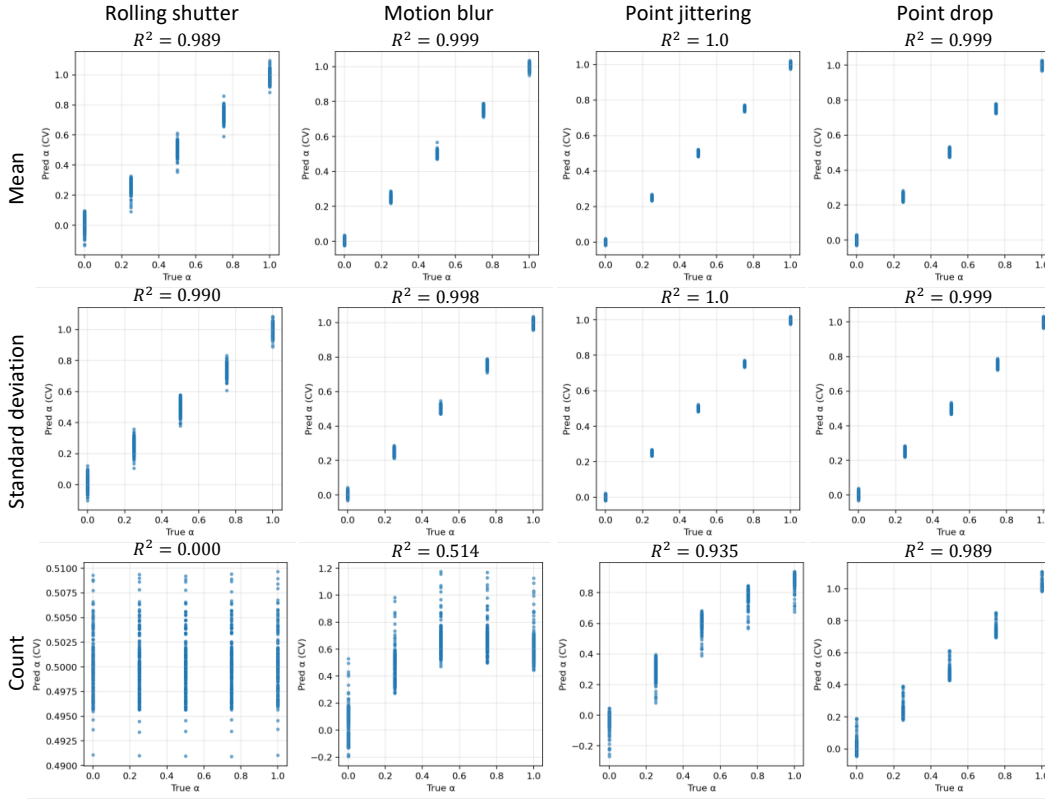


Figure S1: Scatter plots of predicted versus ground-truth perturbation strength ( $\alpha$ ) for four types of synthetic corruptions. The high predictive power of our style embedding features ( $R^2 \approx 1.0$ ) demonstrates that encoder statistics effectively capture the structural domain gap.

mean and standard deviation, effectively capture the point structural domain gap. Given this, the hierarchical entanglement of style embeddings across classes suggests that the domain gap is also entangled along the class hierarchy. This observation provides the rationale for our design choice of a class-hierarchical style embedding.

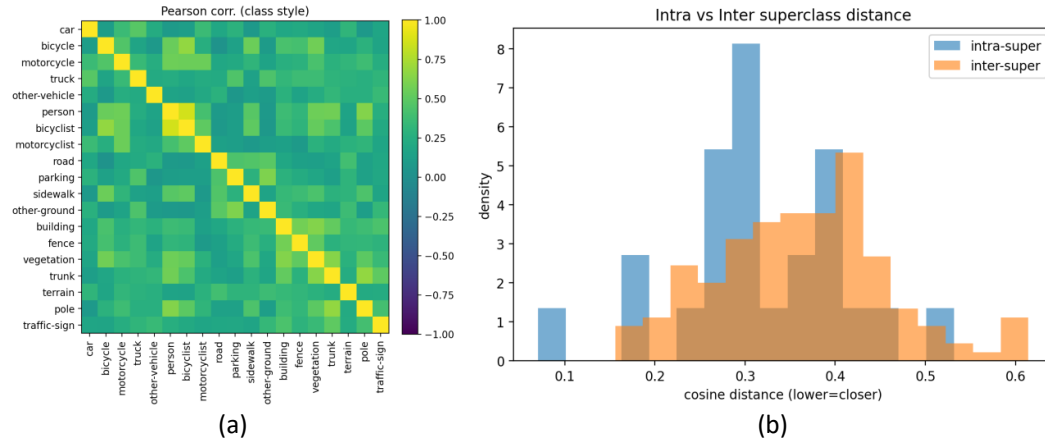


Figure S2: (a) *Pearson correlation* matrix for mean and standard deviation of encoder features. (b) Cosine distance of mean and standard deviation of encoder features.

## B IMPACT OF BINNING METHOD IN CLASS-AGNOSTIC STYLE EMBEDDING

As shown in Table S1, we conducted an experiment by replacing the distance-based binning in the class-agnostic style embedding with a pitch-based binning strategy, following LaserMix (Kong et al., 2023). This modification resulted in a minor performance drop of approximately -0.1 mIoU. The result suggests that the effectiveness of class-agnostic style embedding is not highly sensitive to the choice of binning strategy.

Binning Style	mIoU
Ours	44.7
LaserMix Style	44.6

Table S1: Ablation comparison between the two binning methods for class-agnostic style embedding.

## C IMPACT OF LABEL NOISE FOR CLASSWISE STYLE EMBEDDING

As shown in Tab. S2, we conducted an experiment to examine the impact of pseudo-label noises on classwise style embedding adversarial learning for the target real data. Specifically, we replaced pseudo labels with ground-truth labels of real data when computing the style embeddings for adversarial training. Interestingly, we observed no performance improvement under this setting. This is because style embedding-based adversarial learning does not rely on label supervision in the traditional sense. Instead, it functions as a domain cue rather than a label prediction mechanism. Therefore, even when pseudo labels contain errors, they do not introduce harmful learning signals, and the training behaves similarly to scene-wise adversarial learning, which is class-agnostic.

## D MORE STUDY ABOUT CLASS-HIERARCHICAL STYLE EMBEDDING

To assess the sensitivity of our hierarchical style embedding adversarial learning to changes in class hierarchy, we conducted experiments using modified hierarchies. Specifically, we grouped `bicycle` and `motorcycle` under new superclass, *two-wheeler*, and moved `terrain` into the *natural* category. As shown in Table S3, this modification resulted in only a minor performance drop of approximately -0.1 mIoU. We further experimented with a coarser hierarchy that separates classes into only two groups: *things* and *stuff*. This setting led to a drop of -0.1 mIoU. These small performance degradations indicate that our method is not highly sensitive to the choice of class hierarchy.

## E HYPERPARAMETER ABLATIONS

To evaluate parameter sensitivity, we conducted experiments on: 1) class-agnostic maximum range, 2) class-agnostic binning range, 3) self-training threshold, 4) adversarial loss weight, and 5) intensity self-training loss weight. Reducing the class-agnostic maximum range to 10 yielded 44.7 mIoU, while increasing it to 60 resulted in 44.8 mIoU; thus, performance was not highly sensitive to maximum range. Adjusting the class-agnostic binning range to 2 resulted in 44.7 mIoU, whereas increasing it to 10 achieved 44.8 mIoU, indicating minor sensitivity to this hyperparameter. Lowering the self-training threshold to 0.2 produced 44.4 mIoU, while raising it to 0.9 resulted in 44.6 mIoU, showing minimal impact on performance. Reducing the adversarial loss weight to  $1 \times 10^{-4}$  yielded 45.0 mIoU, whereas increasing it to  $1e-2$  resulted in 41.3 mIoU. Similar to previous work (Duesterwald et al., 2019), we found that the adversarial loss is comparatively sensitive to its hyperparameters. Finally, decreasing the intensity of self-training loss weight to 1 achieved 44.4 mIoU, while increasing it to 100 dropped performance to 43.3 mIoU. These results confirm that attempting overly implicit semantic learning through intensity values degrades performance.

Label Setup	mIoU
w/ Pseudo Labels	44.7
w/ GT Labels	44.6

Table S2: Comparison between classwise/class-hierarchical style embedding utilizing pseudo-label training and ground-truth label.

Hierarchy Version	mIoU
Ours	44.7
Edited Hierarchy	44.7
Things-Stuff Only	44.6

Table S3: Ablation comparison with other class-hierarchy formulations.

## F CONFUSION MATRIX

As shown in Fig. S3, the confusion matrix improves notably for dynamic and structurally complex object classes such as `person`, `bicyclist`, and `motorcycle`. These improvements indicate that our method is particularly effective for classes with greater complexity and dynamics, which typically suffer from additional synthetic-to-real domain gaps.

As in the SemanticKITTI experiments, Tab. S4 shows that our method also improves the confusion matrix on the SemanticPOSS dataset, particularly for dynamic or geometrically complex objects such as `bicyclist` and `car`. We also observe performance gains for classes such as `vegetation` and `trunk`.

## G VISUAL & TEXT PROMPTS FOR ATTAINING CLASS HIERARCHY

We provided the following prompt to construct the class hierarchy (see Fig. S5 and Fig. S6). To avoid any influence from color cues in point cloud representations, the input was rendered entirely in black. The results shown in Fig. S6 were conducted on SemanticKITTI, and the same approach was applied to obtain the class hierarchy for SemanticPOSS.

## H MORE QUALITATIVE RESULTS

Qualitative results are also provided: Fig. S7 demonstrates that our method more accurately predicts complex and dynamic classes such as `motorcycle` (row 1), `person` (row 2), and `bicyclist` (rows 3 and 4) in *SynLiDAR-to-SemanticKITTI* benchmark; similarly, Fig. S8 shows improved predictions by our method for `trash-can` (row 1), `vegetation` (row 2), and `trunk` (row 3), and `person` (row 4) in *SynLiDAR-to-SemanticPOSS* benchmark.

## I REPRODUCIBILITY STATEMENT

To validate the reproducibility of our proposed method, we include the implementation code in the supplementary material.

## J USAGE FOR LARGE LANGUAGE MODELS

We acknowledge the use of GPT-5 for assistance with the writing and grammar checking of this paper. The final version of the article was subsequently reviewed and edited by all of the authors.

Hyper-param.	Class-Ag. Max Range			Class-Ag. Bin Range			ST Threshold			Adv Loss Weight			ST Loss Weight		
	15	30	60	2	5	10	0.2	0.5	0.9	$10^{-4}$	$10^{-3}$	$10^{-2}$	1	10	100
mIoU (%)	44.7	44.7	<b>44.8</b>	44.7	44.7	<b>44.8</b>	44.4	<b>44.7</b>	44.6	<b>45.0</b>	44.7	41.3	44.4	<b>44.7</b>	43.3

Table S4: Sensitivity analysis of key hyper-parameters. The best mIoU in each block is highlighted in bold.

```

1 def add_motion_blur(points: np.ndarray, alpha: float, max_shift: float =
2   0.30) -> np.ndarray:
3     pts = points.copy()
4     x, y, z = pts[:, 0], pts[:, 1], pts[:, 2]
5     theta = np.arctan2(y, x)
6     r = np.sqrt(x**2 + y**2 + z**2)
7     r_med = np.median(r) + 1e-6
8     tx, ty = -np.sin(theta), np.cos(theta)
9     L = alpha * max_shift * (0.5 + 0.5 * (r / r_med))
10    noise = np.random.uniform(0.3, 1.0, size=r.shape)
11    pts[:, 0] += tx * L * noise
12    pts[:, 1] += ty * L * noise
13    return pts
14
15 def add_rolling_shutter(points: np.ndarray, alpha: float, max_shear:
16   float = 0.20) -> np.ndarray:
17     pts = points.copy()
18     x, y, z = pts[:, 0], pts[:, 1], pts[:, 2]
19     theta = np.arctan2(y, x)
20     s = alpha * max_shear
21     pts[:, 0] = x + s * z * np.sin(theta)
22     pts[:, 1] = y + s * z * np.cos(theta)
23     return pts
24
25 def add_point_drop(points: np.ndarray, alpha: float, max_drop: float =
26   0.5) -> np.ndarray:
27     drop_rate = float(np.clip(alpha * max_drop, 0.0, 0.95))
28     N = points.shape[0]
29     keep = np.random.rand(N) > drop_rate
30     if keep.sum() < 100:
31         keep[np.random.choice(N, size=min(100, N), replace=False)] = True
32     return points[keep]
33
34 def add_jitter(points: np.ndarray, alpha: float, max_sigma: float = 0.05)
35   -> np.ndarray:
36     sigma = alpha * max_sigma
37     noise = np.random.normal(scale=sigma, size=points.shape).astype(np.
38       float32)
39     return points + noise

```

Listing S1: Python code for synthetic data perturbation.

## REFERENCES

- Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3075–3084, 2019.
- Evelyn Duesterwald, Anupama Murthi, Ganesh Venkataraman, Mathieu Sinn, and Deepak Vijaykeerthy. Exploring the hyperparameter landscape of adversarial robustness. *arXiv preprint arXiv:1905.03837*, 2019.
- Lingdong Kong, Jiawei Ren, Liang Pan, and Ziwei Liu. Lasermix for semi-supervised lidar semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern*

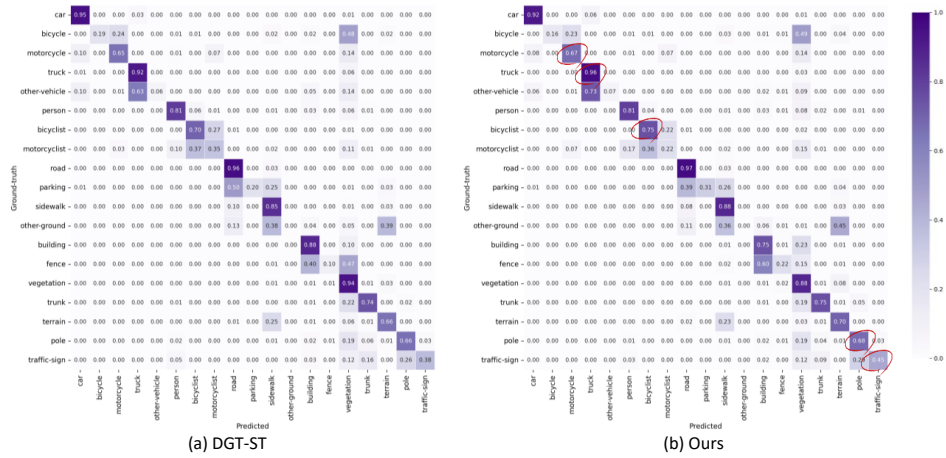


Figure S3: Confusion matrix of (a) DGT-ST and (b) Ours, validated on *SynLiDAR*  $\rightarrow$  *SemanticKITTI* benchmark.

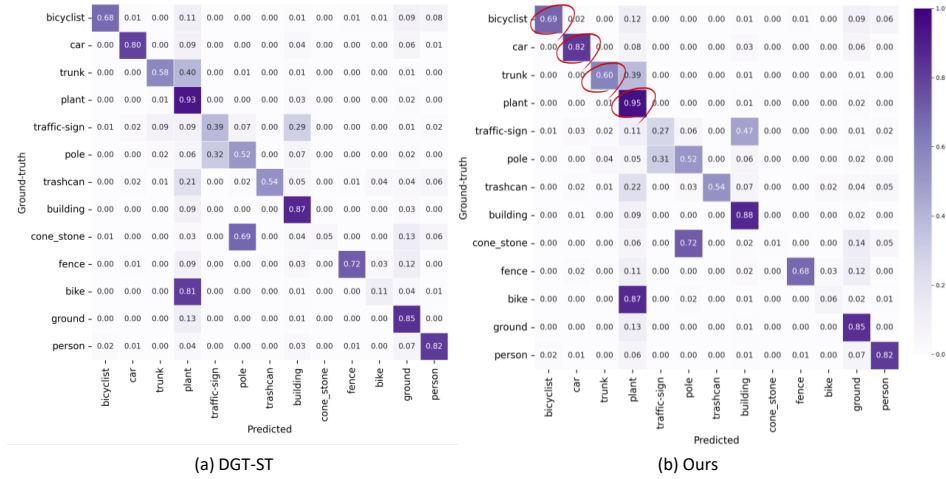


Figure S4: Confusion matrix of (a) DGT-ST and (b) Ours, validated on *SynLiDAR*  $\rightarrow$  *SemanticPOSS* benchmark.

*Recognition*, pp. 21705–21715, 2023.

324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377

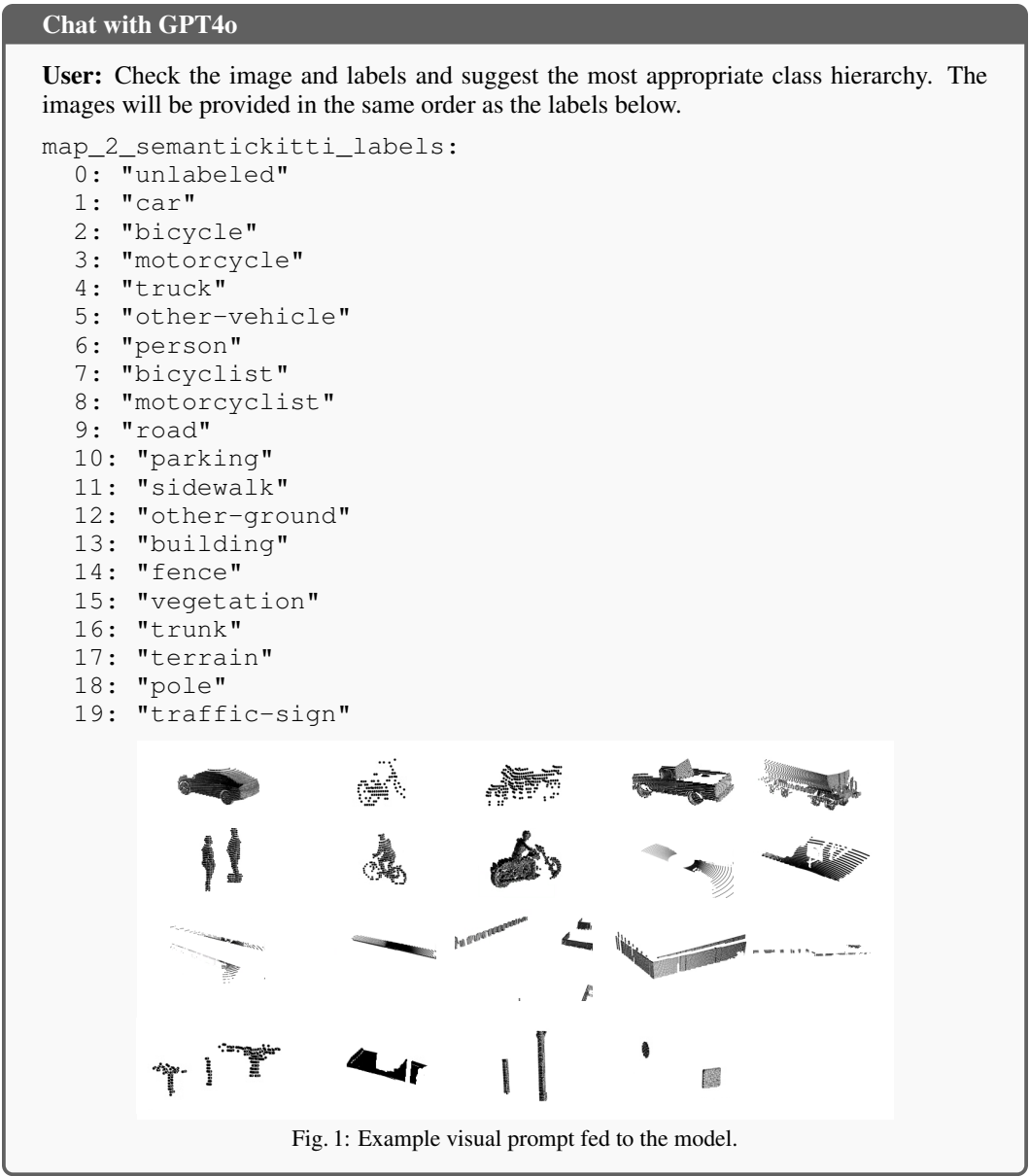


Figure S5: Visual-text prompt supplied to GPT-4o to compute the class hierarchy.

378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431

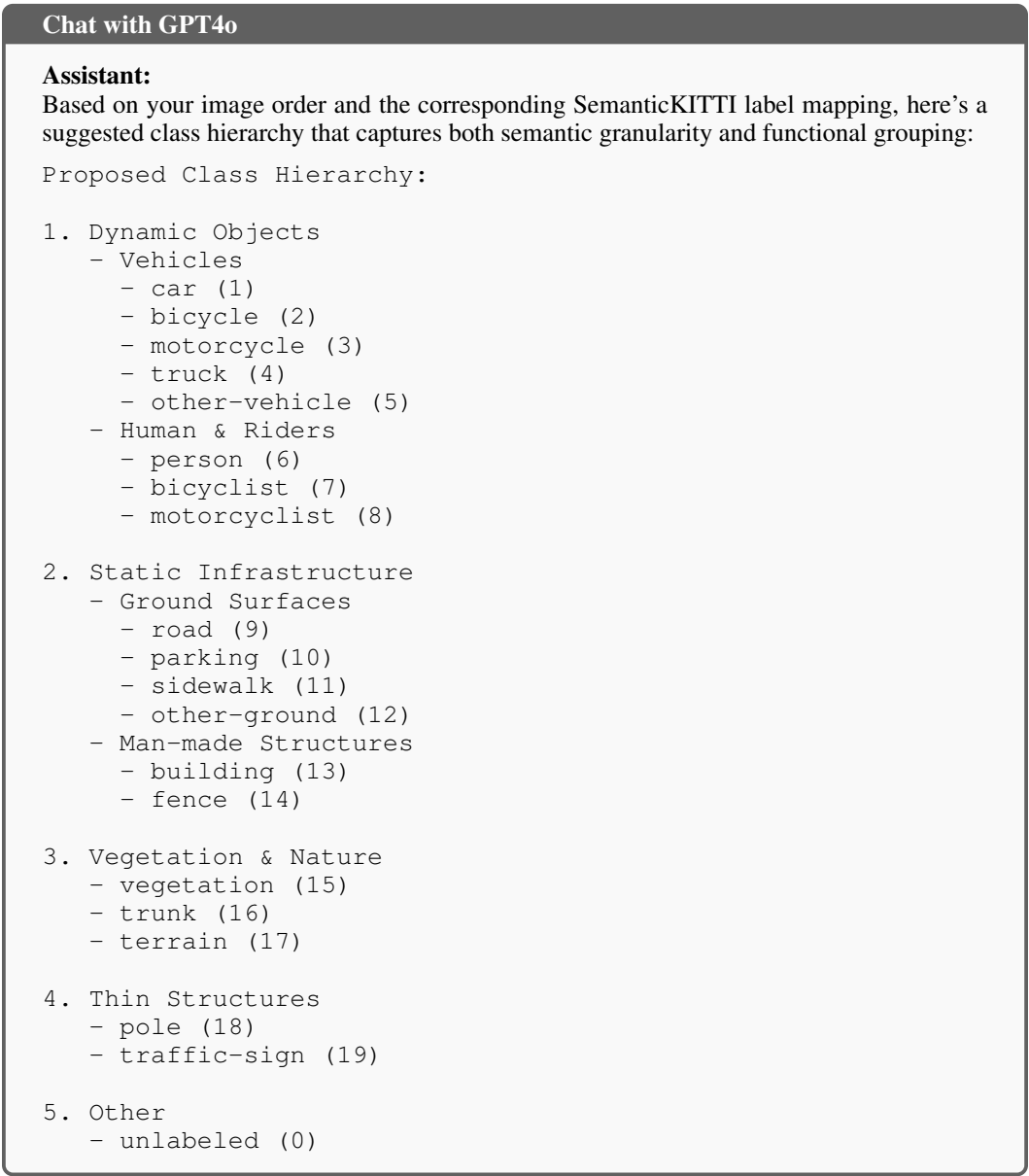


Figure S6: Response of GPT-4o for visual-text prompt.



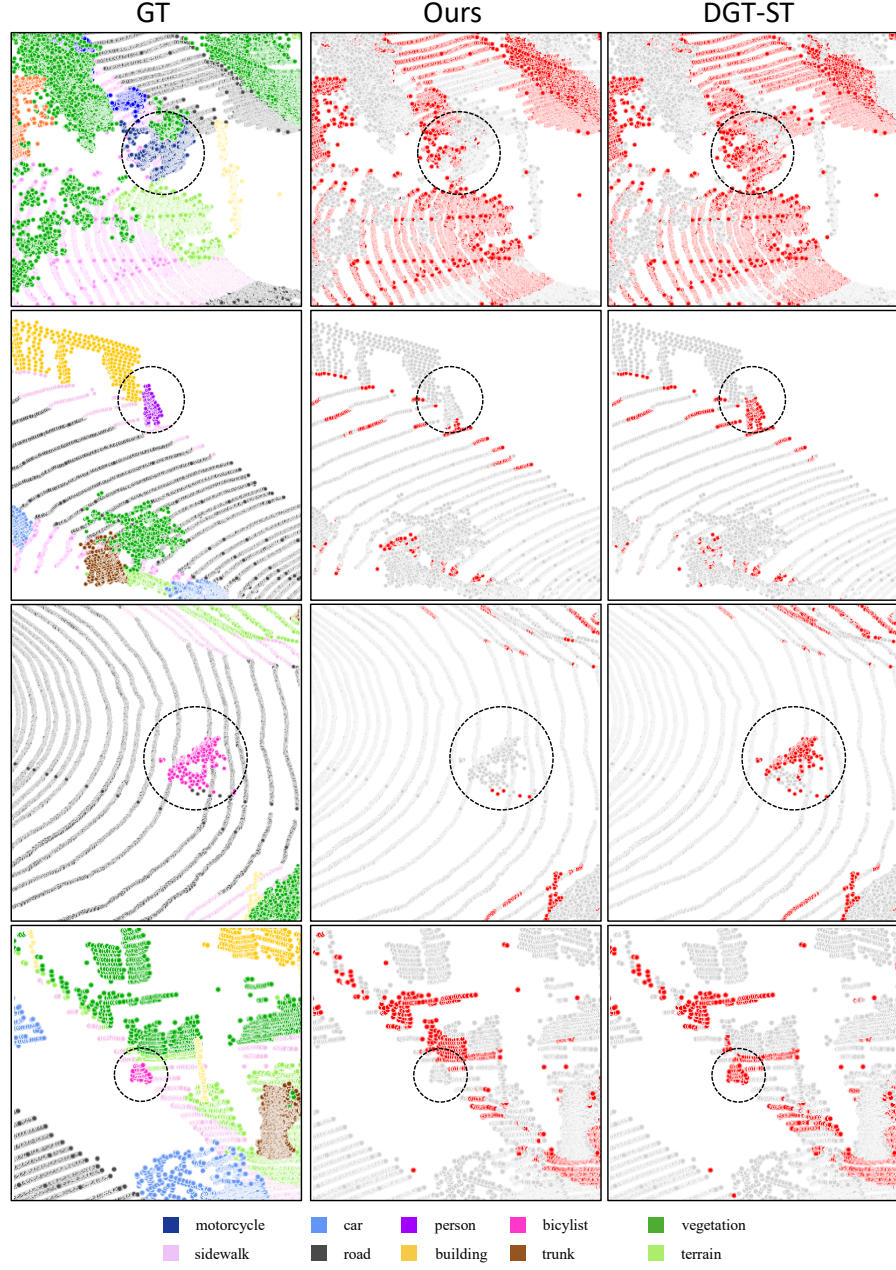


Figure S7: Qualitative results on *SynLiDAR*  $\rightarrow$  *SemanticKITTI* benchmark. Red=incorrect, Gray=correct.

486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539

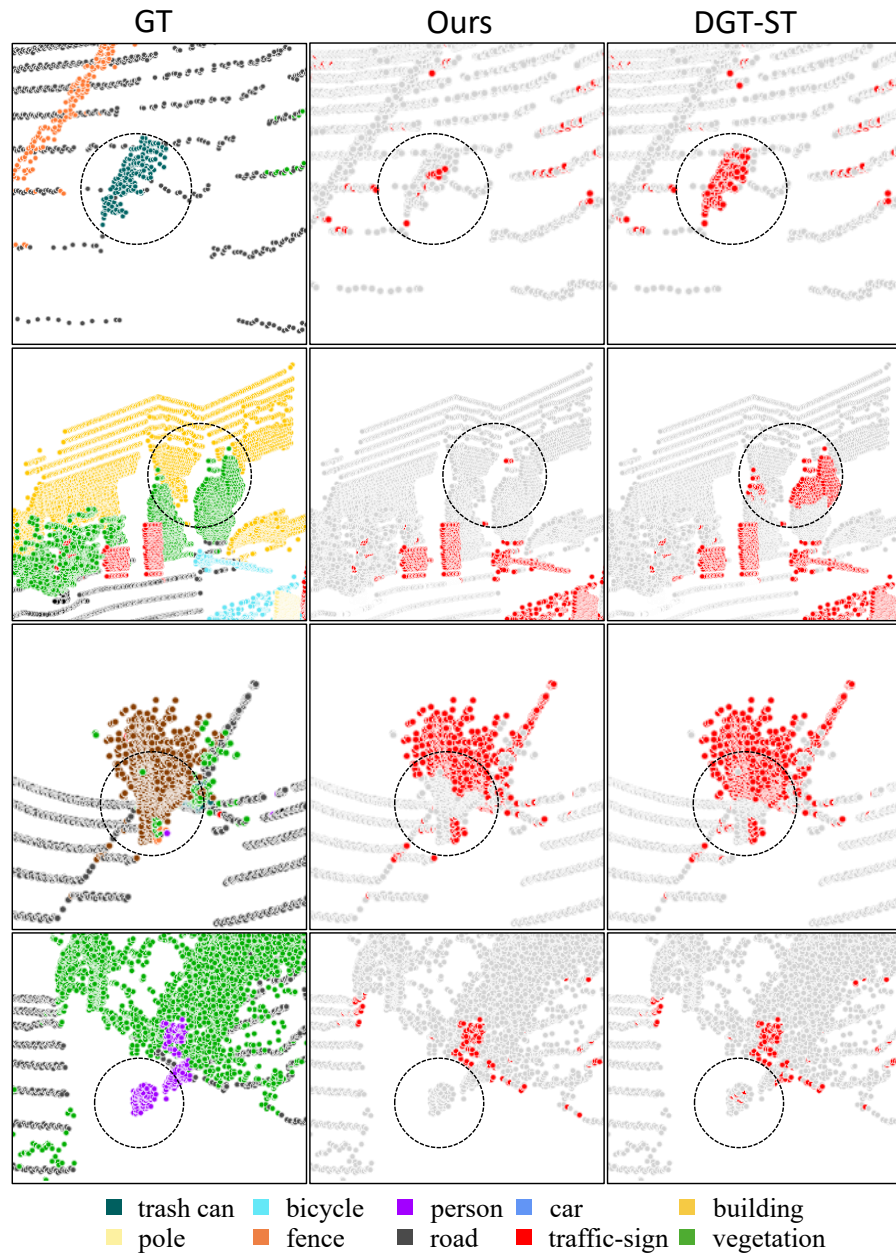


Figure S8: Qualitative results on *SynLiDAR*  $\rightarrow$  *SemanticPOSS* benchmark. Red=incorrect, Gray=correct.