Supplementary Material

We provide more information regarding our flow estimation method SCOOP. Section A presents the derivation of point cloud correspondence as an optimal transport problem and the solution by the Sinkhorn algorithm. Section B includes additional results for the experiments presented in the paper. In Section C, we report the results of an additional experiment on a non-occluded data version. Finally, section D elaborates on our implementation details, including network architecture, training and inference procedure, and the optimization settings of SCOOP.

A. Correspondence as Optimal Transport

As mentioned in the paper, our correspondence-based flow between the point clouds $X, Y \in \mathbb{R}^{n \times 3}$ builds on the optimal transport formulation presented in FLOT [18]. For completeness, we briefly review the optimal transport problem and the Sinkhorn algorithm for solving it.

We begin with a hypothetical perfect case, where each source point $x_i \in X$ has an exact matching target point $y_j \in Y$. Thus, the flow field holds:

$$X + F^* = \Pi Y, \tag{16}$$

where $\Pi \in \{0,1\}^{n \times n}$ is a permutation matrix representing the correspondence between the point clouds, with $\Pi_{ij} = 1$ if x_i matches y_j and $\Pi_{ij} = 0$ otherwise.

In this case, estimating the point correspondences can be modeled as an optimal transport problem [18]. Assuming that each point in X has a mass $\frac{1}{n}$ and each point in Y receives a mass $\frac{1}{n}$, the optimal mass transport is given by:

$$\begin{aligned} T^* = &\operatorname*{argmin}_{T \in \mathbb{R}_+^{n \times n}} \sum_{ij} C_{ij} T_{ij} \\ &\text{such that} \quad T 1_n = \frac{1}{n} 1_n, \quad T^\top 1_n = \frac{1}{n} 1_n, \end{aligned} \tag{17}$$

where $1_n \in \mathbb{R}^n$ is a vector with all entries equal 1, $C_{ij} \geq 0$ is the transport cost from point x_i to point y_j, and $T_{ij} \geq 0$ is the amount of mass transported between these points. The two terms on the second row of Equation 17 are mass constraints, demanding that the total mass delivered from each source point and received by each target point is exactly $\frac{1}{n}$. T^* is optimal in the sense that the mass is transported from X to Y with minimal cost.

In practice, usually, there is no perfect match between the point clouds due to objects appearing in or disappearing from the scene or different points sampled on the scene's surface, and the mass constraints in Equation 17 do not hold. Thus, instead of Equation 17, we used the relaxed version of the transport problem presented in Equation 3 in the paper. The relaxed transport problem is solved by the Sinkhorn algorithm [2, 3], which estimates T^* from C. We provide the algorithm's details in Algorithm 1. In our implementation, the number of iterations M is set to 1.

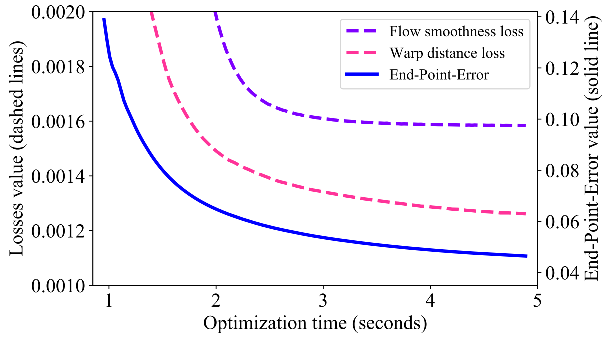

Figure 8. **Refinement evolution.** SCOOP was trained on FT3D$_o$ and evaluated on KITTI$_o$. We present the refinement loss values and the corresponding End-Point-Error (EPE) during the optimization process. The losses are effectively minimized and result in a substantial reduction of the flow estimation error.

Method	Refinement	$EPE\downarrow$	$AS\uparrow$	$AR\uparrow$	$Out.\downarrow$
FLOT [18]	✗	0.142	30.6	61.9	57.6
FLOT [18]	✓	**0.048**	**89.0**	**93.5**	**20.4**
SCOOP$^+$ (ours)	✗	0.139	36.1	63.6	54.9
SCOOP$^+$ (ours)	✓	**0.047**	**91.3**	**95.0**	**18.6**

Table 3. **Our refinement optimization for another method.** FLOT and SCOOP were trained on 1,800 examples from FT3D$_o$ and tested on KITTI$_o$, without or with our flow refinement component. The proposed refinement module considerably improves the flow estimation performance for both methods.

B. Additional Results

B.1. Refinement Evolution

We examine the relationship between our self-supervised losses in the flow refinement process, given in Equation 14, and the resulting End-Point-Error metric (EPE), defined in subsection 4.1. Figure 8 shows the results (for better visualization, we multiply the smoothness loss value by a factor of $4 \cdot 10^{-2}$). During run-time, we minimize our smoothness and distance losses without using ground-truth flow labels. As a byproduct, the EPE is reduced as well. This experiment implies that our refinement objective in Equation 14 correlates with the flow estimation error and serves as a good proxy for its minimization.

B.2. Refinement Optimization for Another Method

A natural question is whether a flow estimation method other than ours can benefit from the proposed refinement optimization module. To address this question, we trained

Algorithm 1: The Sinkhorn Algorithm.

```
Data: cost matrix $C$, parameters $\epsilon, \lambda \geq 0,\ M > 0$.
Result: optimal transport matrix $T^*$.
$T \leftarrow \exp(-C/\epsilon)$;
$a \leftarrow \frac{1}{n}1_n$;
for $m = 1, \ldots, M$ do
    $b \leftarrow (\frac{1}{n}1_n/(T^\top a))^{\lambda/(\lambda+\epsilon)}$;
    $a \leftarrow (\frac{1}{n}1_n/(Tb))^{\lambda/(\lambda+\epsilon)}$;
end
$T^* \leftarrow \mathrm{diag}(a)\, T\, \mathrm{diag}(b)$;
```

FLOT [18] on 1,800 examples from the FT3D$_o$ train set, as done for our method. Then, we evaluated FLOT's performance on the KITTI$_o$ data without or with our run-time refinement (with correspondence confidence equal to 1 for all the source points). Table 3 summarizes the results.

Training on a 10% fraction of FT3D$_o$ data degrades FLOT's performance in comparison to using the complete dataset, as reported in Table 1 in the main body. However, our refinement optimization substantially contributes to the flow precision of FLOT and even yields better results compared to using the whole training set. This experiment hints that our proposed run-time refinement is not tailor-made for SCOOP and can benefit another method as well.

B.3. Qualitative Results

In Figure 9, we present additional results of SCOOP for KITTI$_o$ data for various challenging cases. For example, our method can gracefully handle different point densities, as cars with varying distances from the LiDAR sensor exhibit. In addition, since we require consistency of the flow field over the point cloud, SCOOP can correctly estimate the flow for an object with a repetitive structure, such as a fence. At the same time, our flow estimation method is versatile. It copes with shapes of different geometry and size, such as the pole and the facade. SCOOP can also predict translation vectors of different directions and magnitudes, as for the car and pole.

B.4. Ablation Runs

In Table 4, we report results of our method for different train set sizes of FT3D$_o$. The table shows that a 10% fraction of the FT3D$_o$ data is sufficient for SCOOP to converge to its optimal performance.

Table 5 presents additional ablation experiments. In this round, we examined the following settings (one configuration change at a time). (a) Turn off the Sinkhorn normalization. In this case, we used $T = \exp(-C/\epsilon)$ instead of T^* from Algorithm 1, and the correspondence construction in Equations 4 and 5 was done with target points with minimal matching cost C rather than maximal transport T^*. (b) Apply a higher number of iterations in the Sinkhorn algorithm by setting $M = 3$ instead of $M = 1$. (c) Linear normalization for the correspondence confidence $p_{x_i} = (s_{x_i} + 1)/2$ instead of the non-linear truncation $p_{x_i} = \max(s_{x_i}, 0)$. In all these settings, the difference in the method's performance was small, implying its robustness to such configuration changes.

FT3D$_o$ number of training examples	180	1,800	18,000
KITTI$_o$ $EPE\downarrow$	0.057	0.047	0.047

Table 4. **Train set size ablation.** We trained SCOOP on the FT3D$_o$ dataset using a different number of instances and measured the EPE on the KITTI$_o$ dataset. A subset of only 1,800 training examples is sufficient for our technique.

Setting	$EPE\downarrow$	$AS\uparrow$	$AR\uparrow$	$Out.\downarrow$
(a) W/O Sinkhorn	0.042	91.6	95.9	16.1
(b) 3 Sinkhorn iterations	0.040	92.9	96.4	15.3
(c) Linear p_{x_i} normalization	0.040	93.5	96.4	15.5
The proposed method	**0.039**	**93.6**	**96.5**	**15.2**

Table 5. **Additional ablations.** We trained SCOOP with different configurations on KITTI$_v$ and evaluated its performance on KITTI$_t$. The table shows that our method is robust to these configuration variations. Details about the ablative settings appear in subsection B.4.

B.5. Limitation

A failure case of SCOOP is presented in Figure 11. When a part of the source scene is completely missing from the target, the correspondence to existing target points is inaccurate, and the flow predicted by our method does not represent the motion of that part. In future work, we plan to detect such wrong matches by remaining inconsistencies in the flow field and leverage the global motion of the scene to deduce the flow for completely occluded regions.

C. An Additional Experiment

In addition to FT3D$_o$ and KITTI$_o$, Gu *et al.* [6] prepared another point cloud version of the FlyingThings3D and KITTI datasets, denoted as FT3D$_s$ and KITTI$_s$, respectively. In their version, all occluded points are removed, and each source point has a matched target point. This version of the datasets is also popular in the scene flow literature, and for a comprehensive evaluation, we report our method's results for this case as well. Additional details about the datasets appear in subsection D.2.

Since the point clouds produced by Gu *et al.* have no occlusions, we adapt our method to the nature of this data. Instead of the distance loss from Equation 10, we use the

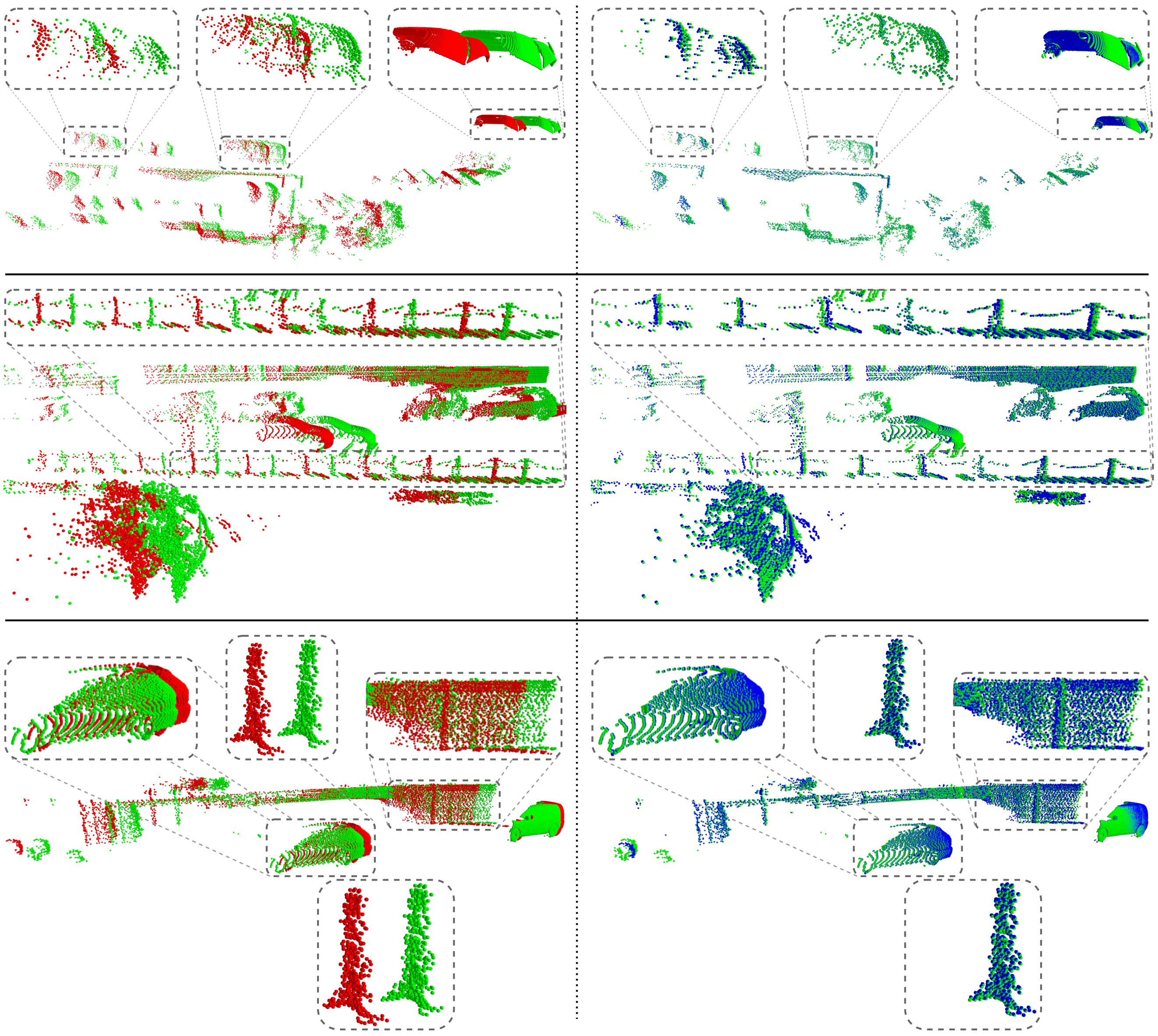

Figure 9. **Visual results.** We applied SCOOP to different LiDAR scenes. The source and target input point clouds are presented in red and green, respectively, and the warped source is shown in blue. Our method is able to predict the scene flow in a variety of challenging scenarios, such as varied point cloud density (top), repetitive structures (middle), and objects with different sizes and motions (bottom).

bidirectional Chamfer Distance loss [9, 23]:

$$\mathcal{L}_{cd} = CD(\widehat{Y}, Y) = \frac{1}{|\widehat{Y}|}\sum_{\hat{y}\in\widehat{Y}} \min_{y\in Y} ||\hat{y} - y||_2^2 + \frac{1}{|Y|}\sum_{y\in Y} \min_{\hat{y}\in\widehat{Y}} ||y - \hat{y}||_2^2, \tag{18}$$

where $\widehat{Y}$ is the softy corresponding point cloud to the source point cloud X (from Equation 7), and Y is the target point cloud. The Chamfer Distance is also used in the refinement process and replaces the first term in the optimization objective in Equation 14. If we define $\widehat{Y}_r = \widehat{Y} + R$, the updated distance loss term for the flow refinement optimization is $CD(\widehat{Y}_r, Y)$. The rest of our method's formulation remains the same.

Following the evaluation protocol of previous work [6, 8, 18], we train SCOOP on FT3D$_\text{s}$ and evaluate the performance on the test set of FT3D$_\text{s}$ and on the KITTI$_\text{s}$ data. Different from prior work, we use only 10% of the FT3D$_\text{s}$ training data, which suffices for our correspondence model

Method	Sup.	$EPE\downarrow$	$AS\uparrow$	$AR\uparrow$	$Out.\downarrow$
FlowNet3D [15]	*Full*	0.114	41.3	77.1	60.2
HPLFlowNet [6]	*Full*	0.080	61.4	85.6	42.9
PointPWC-Net [23]	*Full*	0.059	73.8	92.8	34.2
FLOT [18]	*Full*	0.052	73.2	92.7	35.7
PV-RAFT [22]	*Full*	0.046	81.7	95.7	29.4
FlowStep3D [8]	*Full*	0.046	81.6	96.1	21.7
HCRF-Flow [11]	*Full*	0.049	83.4	95.1	26.1
RCP [5]	*Full*	0.040	85.7	96.4	19.8
Rigid3DSceneFlow [4]	*Full*	0.052	74.6	93.6	36.1
3D-OGFlow [17]	*Full*	0.036	87.9	-	19.7
SCTN [10]	*Full*	0.038	84.7	96.8	26.8
3DFlow [21]	*Full*	**0.028**	**92.9**	**98.2**	14.6
Bi-PointFlowNet [1]	*Full*	**0.028**	91.8	97.8	**14.3**
Ego-motion [20]	*Self*	0.170	25.3	55.0	80.5
PointPWC-Net [23]	*Self*	0.121	32.4	67.4	68.8
Self-Point-Flow [12]	*Self*	0.101	42.3	77.5	60.6
FlowStep3D [8]	*Self*	0.085	53.6	82.6	42.0
RSFNet [7]	*Self*	0.075	58.9	86.2	47.0
RCP [5]	*Self*	0.077	58.6	86.0	41.4
RigidFlow [13]	*Self*	0.069	59.6	87.1	46.4
SCOOP (ours)	*Self*	0.084	56.7	85.1	48.5

Table 6. **Quantitative comparison on the FT3D$_s$ test set.** All the methods were trained on the train split of FT3D$_s$. Our method is on par with other self-supervised methods.

Method	Sup.	$EPE\downarrow$	$AS\uparrow$	$AR\uparrow$	$Out.\downarrow$
FlowNet3D [15]	*Full*	0.177	37.4	66.8	52.7
HPLFlowNet [6]	*Full*	0.117	47.8	77.8	41.0
PointPWC-Net [23]	*Full*	0.069	72.8	88.8	26.5
FLOT [18]	*Full*	0.056	75.5	90.8	24.2
PV-RAFT [22]	*Full*	0.056	82.3	93.7	21.6
FlowStep3D [8]	*Full*	0.055	80.5	92.5	14.9
HCRF-Flow [11]	*Full*	0.053	86.3	94.4	18.0
RCP [5]	*Full*	0.048	84.9	94.5	12.3
Rigid3DSceneFlow [4]	*Full*	0.042	84.9	95.9	20.8
3D-OGFlow [17]	*Full*	0.039	88.2	-	17.5
SCTN [10]	*Full*	0.037	87.3	95.9	17.9
3DFlow [21]	*Full*	0.031	90.5	95.8	16.1
Bi-PointFlowNet [1]	*Full*	0.030	92.0	96.0	14.1
Ego-motion [20]	*Self*	0.415	22.1	37.2	81.0
PointPWC-Net [23]	*Self*	0.255	23.8	49.6	68.6
Self-Point-Flow [12]	*Self*	0.112	52.8	79.4	40.9
FlowStep3D [8]	*Self*	0.102	70.8	83.9	24.6
RSFNet [7]	*Self*	0.092	74.7	87.0	28.3
RCP [5]	*Self*	0.076	78.6	89.2	18.5
RigidFlow [13]	*Self*	0.062	72.4	89.2	26.2
SCOOP (ours)	*Self*	**0.019**	**97.1**	**98.5**	**10.7**

Table 7. **Quantitative comparison on the KITTI$_s$ data.** All the methods were trained on the train split of FT3D$_s$. SCOOP outperforms all the compared alternatives, both the self-supervised and the fully-supervised ones.

to coverage. The evaluation metrics are the same as those in the main body, detailed in subsection 4.1. For both training and testing, we use point clouds with $n = 8192$ points.

Tables 6 and 7 present our test results for FT3D$_s$ and KITTI$_s$, respectively, compared to abundant recent alternative methods. While trained only on a 10% fraction of the data, SCOOP achieves competitive results compared to other self-supervised methods on FT3D$_s$. On the KITTI$_s$ dataset, we surpass the performance of both self and fully-supervised methods for all the evaluation metrics. For example, SCOOP improves the EPE metric by 37% over the very recent Bi-PointFlowNet work [1], reducing the flow estimation error from 0.030 to 0.019 meters. These results suggest that our method is highly effective for the real-world KITTI$_s$ data.

D. Implementation Details

D.1. Network Architecture

The point feature extraction is done by a neural network based on the PointNet++ architecture [19]. The network includes 3 set-convolution layers, which increase the feature channels per point. Each layer contains a multi-layer perceptron, interleaved with instance normalization and a leaky ReLU activation with a negative slope of -0.1. After each convolutional layer, the point features are aggregated by a

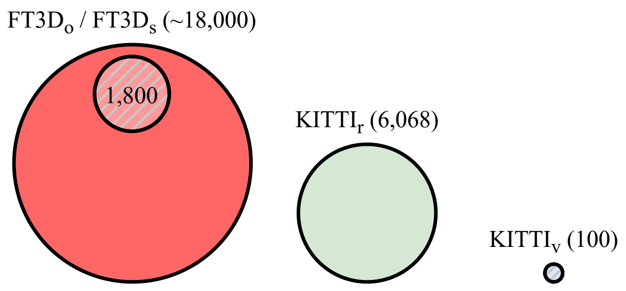

Figure 10. **Visual illustration of the training datasets' size.** We use a small amount of data for training (stripe pattern) compared to the amount used by others (solid pattern).

max pooling operation from 32 Euclidean nearest neighbor points. The coordinate difference between the point and its neighbors is concatenated to the input features of every set-convolution layer. Table 8 details the feature dimensions of the network's layers.

D.2. Training and Inference

Training dataset size. We illustrate the size of the training datasets in Figure 10. As explained in the paper (subsection 4.2), SCOOP is a data-light method that requires much less training data than other learning-based methods.

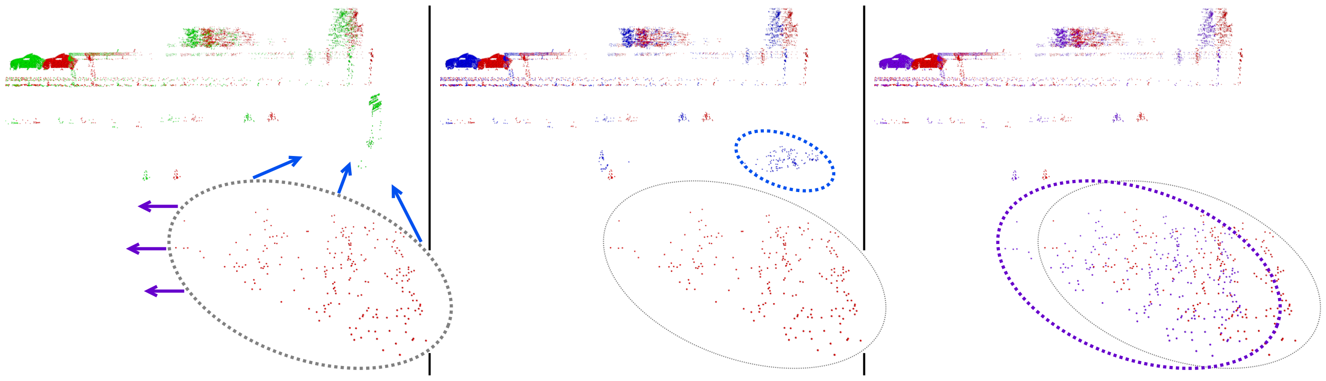

Figure 11. **A failure example.** We show the source point cloud in red, the target in green (left), the translated source by SCOOP in blue (middle), and the translated source by the ground-truth flow in purple (right). A set of source points whose target is completely occluded is marked with a gray ellipse. Its warp by the estimated and the ground-truth flow is delineated by a blue ellipse and a purple ellipse, respectively. Our method struggles to predict the correct flow in such a case.

Network architecture
$concat$(coordinates (3), neighbors' coordinate difference (3))
$SetConv(32, 32, 32)$
$neighbors\ max\ pooling$ (32)
$concat$(features (32), neighbors' coordinate difference (3))
$SetConv(64, 64, 64)$
$neighbors\ max\ pooling$ (64)
$concat$(features (64), neighbors' coordinate difference (3))
$SetConv(128, 128, 128)$
$neighbors\ max\ pooling$ (128)

Table 8. **The architecture of the feature extraction model.** The values in parentheses indicate the per-point feature dimension at each network stage. $concat$ represents a concatenation operation. The coordinate difference and max pooling operation are computed with a neighborhood of 32 nearest points in the Euclidean space. $SetConv$ is the set convolution described in subsection D.1, where the numbers in its parentheses refer to the filter sizes of the multi-layer perceptron.

Train/Test data (#points)	k_f	λ_{flow}	Gradient steps	Update rate
FT3D$_o$/KITTI$_o$ (2,048)	32	1.0	1000	0.05
KITTI$_v$/KITTI$_t$ (2,048)	32	1.0	1000	0.05
FT3D$_o$/KITTI$_o$ (29,951)	32	1.0	150	0.2
KITTI$_v$/KITTI$_t$ (30,814)	32	1.0	150	0.2
FT3D$_s$/FT3D$_s$ (8,192)	16	1.0	1000	0.1
FT3D$_s$/KITTI$_s$ (8,192)	32	1.0	1000	0.05

Table 9. **Refinement hyperparameters.** The table details the values we used for our flow refinement optimization process for different dataset settings. For each setting, we indicate the train/test datasets and the average number of points in the test point clouds.

Occluded data version. The FT3D$_o$ dataset contains point clouds of 8,192 points, where the z-axis coincides with the depth axis, and the maximal z-value is limited to 35 meters [15]. In the KITTI$_o$ dataset, there are several tens of thousands of points per scene, with a different number of points for the source and target point clouds, denoted as N_s and N_t, respectively. We align the z-axis of KITTI$_o$ to the depth axis and trim the maximal z-value to 35 meters, as done for the FT3D$_o$ data [18].

For memory-efficient training, SCOOP is trained on point sets with the same number of $n =$ 2,048 points sampled at random from the original point clouds. Following previous work [13,15,16,18], we evaluate SCOOP on small test point clouds of randomly sampled 2,048 points. However, we also employ our method to infer the flow for all the points N_s in the source point cloud, as explained next.

At the test-time, we randomly shuffle the source points and the target points, divide them into disjoint chunks of $n =$ 2,048 points, and compute the point features Φ_X and Φ_Y for each chunk, as done in the training stage. If the number of points is not divided by n, we pad with randomly selected points from within the point cloud to the closest multiple of n. Then, for each source chuck, we calculate the matching cost with respect to all the points in the target, obtain a cost matrix $C_{chunk} \in \mathbb{R}^{n \times N_t}$, and compute the correspondence-based flow $F_{chunk} \in \mathbb{R}^{n \times 3}$. Afterward, we collect the flow from the different chunks, remove the padded points (if any), and get the per-point flow $F \in \mathbb{R}^{N_s \times 3}$.

Our inference process for the complete point clouds has several advantages. First, it can be used for source and target point clouds with different cardinality since each point cloud is padded to a multiple of n. Second, as we extract point features in chunks of n points, the process remains memory-efficient and emulates inputs to the network similar to the training phase. Third, it utilizes the complete point information from the target by computing the cost matrix and correspondence flow at the original target point cloud resolution.

Similarly, we perform the flow refinement optimization at the full source and target point cloud resolution. Namely, the distance loss for flow refinement is computed between the complete warped source and the complete target, and the flow smoothness loss is calculated at the original source point cloud resolution. This way, the whole scene data is exploited.

For network-only baselines [12, 13, 16], inferring the scene flow directly for the high point cloud resolution is computationally infeasible, let alone training the models on the complete large point clouds. Thus, following their training scheme on small point clouds with 2,048 points, we divided the original point clouds into chunks of 2,048 points, applied the models, and averaged the results across the chunks to obtain the evaluation for all the points in the dataset.

We note that our results for Neural Prior [14] are different from those reported in their paper. In their work, they did not limit the depth value of the point clouds. However, in our work, we used points with a maximal depth of 35 meters to align with previous learning-based methods [15, 18].

Non-occluded data version. The FT3D$_\text{s}$ dataset has 19,640 and 3,824 point cloud pairs for the train and test sets, respectively. Each point cloud has 8,192 points. We keep aside 2,000 examples from the training set for validation during training. The KITTI$_\text{s}$ data include 200 pairs of source and target point clouds, where 142 of which are used for evaluation. Ground points are removed by a threshold on the height. In both datasets, points with a depth larger than 35 meters are excluded, as done by Gu *et al.* [6]. For testing, we randomly sample 8,192 points from the source and target point clouds each. Our inference time for FT3D$_\text{s}$ and KITTI$_\text{s}$ is about 3.7 seconds.

D.3. Optimization

We trained our point embedding model with an ADAM optimizer with an initial learning rate of 0.001 and a momentum of 0.9. On the FT3D$_\text{o}$ dataset, we trained the model for 30, 100, and 200 epochs when using 180, 1,800, and 18,000 training examples, respectively. For training on the KITTI$_\text{v}$ dataset, we used 400 epochs, and the learning rate was reduced by a factor of 10 after 340 epochs. In all these cases, the batch size was 4. For the FT3D$_\text{s}$ dataset, we selected 1,800 examples at random and trained our model for 60 epochs with a batch size of 1. The learning rate was multiplied by 0.1 after 50 epochs.

As mentioned in the paper, we optimized ϵ and λ from the regularized transport problem (Equation 3 in the main body) during the training process. Their $\log$ value was learned to ensure their non-negativity. In addition, we added a constant of 0.03 to the learned value of ϵ for the numerical stability of the learning process.

The refinement component R^* in Equation 14 in the paper was defined as an optimizable variable and initialized to a matrix of zeros. We optimized its value using an ADAM optimizer with a momentum of 0.9. Further hyperparameters are given in Table 9. All our experiments were done on an NVIDIA Titan Xp GPU.

References

[1] Wencan Cheng and Jong Hwan Ko. Bi-PointFlowNet: Bidirectional Learning for Point Cloud Based Scene Flow Estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 108–124, 2022. 1, 2, 6, 14

[2] Lenaic Chizat, Gabriel Peyré, Bernhard Schmitzer, and François-Xavier Vialard. Scaling Algorithms for Unbalanced Transport Problems. *Mathematics of Computation*, 87:2563–2609, 2018. 3, 11

[3] Marco Cuturi. Sinkhorn Distances: Lightspeed Computation of Optimal Transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2292–2300, 2013. 3, 11

[4] Zan Gojcic, Or Litany, Andreas Wieser, Leonidas J. Guibas, and Tolga Birdal. Weakly Supervised Learning of Rigid 3D Scene Flow. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5692–5703, 2021. 1, 3, 14

[5] Xiaodong Gu, Chengzhou Tang, Weihao Yuan, Zuozhuo Dai, Siyu Zhu, and Ping Tan. RCP: Recurrent Closest Point for Scene Flow Estimation on 3D Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8216–8226, 2022. 14

[6] Xiuye Gu, Yijie Wang, Chongruo Wu, Yong Jae Lee, and Panqu Wang. HPLFlowNet: Hierarchical Permutohedral Lattice FlowNet for Scene Flow Estimation on Large-Scale Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3254–3263, 2019. 1, 2, 12, 13, 14, 16

[7] Pan He, Patrick Emami, Sanjay Ranka, and Anand Rangarajan. Self-Supervised Robust Scene Flow Estimation via the Alignment of Probability Density Functions. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 861–869, 2022. 14

[8] Yair Kittenplon, Yonina C. Eldar, and Dan Raviv. FlowStep3D: Model Unrolling for Self-Supervised Scene Flow Estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4114–4123, 2021. 1, 3, 4, 13, 14

[9] Itai Lang, Dvir Ginzburg, Shai Avidan, and Dan Raviv. DPC: Unsupervised Deep Point Correspondence via Cross and Self Construction. In *Proceedings of the International Conference on 3D Vision (3DV)*, pages 1442–1451, 2021. 2, 3, 13

[10] Bing Li, Cheng Zheng, Silvio Giancola, and Bernard Ghanem. SCTN: Sparse Convolution-Transformer Network for Scene Flow Estimation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1254–1262, 2022. 14

[11] Ruibo Li, Guosheng Lin, Tong He, Fayao Liu, and Chunhua Shen. HCRF-Flow: Scene Flow from Point Clouds with Continuous High-order CRFs and Position-aware Flow Embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 364–373, 2021. 14

[12] Ruibo Li, Guosheng Lin, and Lihua Xie. Self-Point-Flow: Self-Supervised Scene Flow Estimation from Point Clouds with Optimal Transport and Random Walk. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15577–15586, 2021. 1, 2, 3, 5, 6, 7, 14, 16

[13] Ruibo Li, Chi Zhang, Guosheng Lin, Zhe Wang, and Chunhua Shen. RigidFlow: Self-Supervised Scene Flow Learning on Point Clouds by Local Rigidity Prior. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16959–16968, 2022. 1, 2, 6, 7, 14, 15, 16

[14] Xueqian Li, Jhony Kaesemodel Pontes, and Simon Lucey. Neural Scene Flow Prior. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 7838–7851, 2021. 2, 3, 6, 7, 16

[15] Xingyu Liu, Charles R. Qi, and Leonidas J. Guibas. FlowNet3D: Learning Scene Flow in 3D Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 529–537, 2019. 1, 2, 5, 6, 14, 15, 16

[16] Himangi Mittal, Brian Okorn, and David Held. Just Go with the Flow: Self-Supervised Scene Flow Estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11177–11185, 2020. 1, 2, 4, 5, 6, 7, 15, 16

[17] Bojun Ouyang and Dan Raviv. Occlusion Guided Self-supervised Scene Flow Estimation on 3D Point Clouds. In *Proceedings of the International Conference on 3D Vision (3DV)*, pages 782–791, 2021. 14

[18] Gilles Puy, Alexandre Boulch, and Renaud Marlet. FLOT: Scene Flow on Point Clouds Guided by Optimal Transport. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 527–544, 2020. 1, 2, 3, 4, 5, 6, 7, 11, 12, 13, 14, 15, 16

[19] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5099–5108, 2017. 3, 5, 14

[20] Ivan Tishchenko, Sandro Lombardi, Martin R. Oswald, and Marc Pollefeys. Self-Supervised Learning of Non-Rigid Residual Flow and Ego-Motion. In *Proceedings of the International Conference on 3D Vision (3DV)*, pages 150–159, 2020. 14

[21] Guangming Wang, Yunzhe Hu, Zhe Liu, Yiyang Zhou, Masayoshi Tomizuka, Wei Zhan, and Hesheng Wang. What Matters for 3D Scene Flow Network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 38–55, 2022. 14

[22] Yi Wei, Ziyi Wang, Yongming Rao, Jiwen Lu, and Jie Zhou. PV-RAFT: Point-Voxel Correlation Fields for Scene Flow Estimation of Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6954–6963, 2021. 14

[23] Wenxuan Wu, Zhi Yuan Wang, Zhuwen Li, Wei Liu, and Li Fuxin. PointPWC-Net: Cost Volume on Point Clouds for (Self-) Supervised Scene Flow Estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 88–107, 2020. 1, 2, 13, 14