Uncertainty in LiDAR Scene Flow Estimation

Daniel Jean-Soo Rhee

May 2026

Abstract

In visual perception of autonomous systems, LiDAR Scene Flow is a problem that estimates 3D motion of individual LiDAR points between consecutive point clouds. Most current LiDAR Scene Flow methods rely on sparse voxelization fed into an encoder decoder, and then devoxelization to assign flow vectors to individual points in the point cloud. However, these methods do not provide uncertainty estimates that can be used to calibrate confidence of predictions in life or death situations. This thesis proposes an additional head at the end of the decoder, which takes the decoder output to evaluate the uncertainty of the model's prediction. With β-NLL supervision on a multilayer perceptron added to a pretrained LiDAR Scene Flow network, confidence can be predicted with no or minimal modification to the original network. After scaling, this heteroscedastic approach produces results with low negative log likelihood and has low expected normalized calibration error results, despite the model demonstrating poor understanding of different object classes. To the best of my knowledge, this is the first learned variance uncertainty head applied to LiDAR Scene Flow designed for modern sparse voxelization with encoder decoder approaches.

1 Introduction

Understanding motion in visual perception is necessary in autonomous systems, as it allows for detection of dynamic objects and a better understanding of a system's surroundings. Many modern autonomous vehicle approaches include Light Detection And Ranging (LiDAR) sensors for perception, primarily for accurate distance measurements and mapping. LiDAR sensors produce point clouds, which are collections of 3D points in space, with each point having x, y and z coordinates, typically in the reference frame of the sensor. LiDAR Scene Flow is a motion understanding technique that estimates the 3D motion vectors of individual points in a LiDAR point cloud between consecutive LiDAR sweeps [9]. LiDAR Scene Flow is critical to many autonomous system tasks such as 3D object tracking [22] and multi-object tracking [16], making it critical to robustifying autonomous vehicles.

For autonomous vehicles, understanding the reliability of predictions is just as important as the predictions themselves. Autonomous vehicles should be able to weigh decisions based on the uncertainty in a model's prediction. Simultaneous localization and mapping (SLAM), which is used in traditional path planners, relies on estimating the location of surrounding obstacles based on uncertainty in sensors [1]. Without this understanding, vehicles can make life or death decisions without understanding potential error in their sensors.

Many supervised state of the art methods for LiDAR Scene Flow estimation such as DeltaFlow [23] and UniFlow [8] use voxelization as opposed to per point learning [5] or diffusion [12] and turn regions of points into voxels to estimate the motion of the voxels instead of individual points. To the best of my knowledge, no voxelization based methods produce per point heteroscedastic uncertainty. To this end, I propose a β-NLL (Negative Log Likelihood) supervised head that produces per-point heteroscedastic uncertainty estimates on top of a pretrained backbone. This uncertainty head produces a computationally lightweight addition that can be used to understand the uncertainty of LiDAR Scene Flow predictions.

2 Related Work

LiDAR Scene Flow Estimation is a very well studied problem. FlowNet3D [11] was one of the first notable approaches and extracts point features and then regresses per point motion with many subsequent improvements such as pyramid, warping, and cost

volume [18] later introduced to further improve the method. Alternatively, some approaches such as DifFlow3D [10] utilize diffusion for scene flow and can also be self supervised [12]. Transformer based approaches such as PT-FlowNet [4] also exist, and can perform better in longer scenarios with more point clouds. Transformer based approaches are also more deployment friendly, as they can be compiled to TensorRT, while sparse convolution approaches typically cannot [26]. Most modern state of the art approaches, like DeltaFlow [23] and UniFlow [8] rely on voxelization to learn regions of movement as opposed to individual point learning.

LiDAR Scene Flow challenges, like the Argoverse [21] and Waymo [5] challenges do not contain uncertainty metrics, and instead focus on the accuracy of the predictions themselves. Uncertainty estimation for LiDAR Scene Flow is thus largely unstudied, despite similar uncer problems such as optical flow having much greater study in uncertainty [10] [20]. To my knowledge, only two published methods, DifFlow3D [10] and DiffSF [25], contain a per point uncertainty estimate, and use diffusion sampling rather than learned variance for uncertainty estimation.

DifFlow3D works by doing iterative diffusion based refinement on scene flow residuals and produces per point uncertainty estimations in each refinement [10]. DifFlow3D supervises its uncertainty against the ground truth so the uncertainty decreases as the model converges, making it explicitly trained and produced through iterative refinement. DiffSF works by using a transformer based scene flow backbone to produce predictions, before denoising is done with diffusion [25]. However, uncertainty is largely a side effect of using diffusion. This study's work instead aims to obtain uncertainty from per point variance instead of diffusion or iterative sampling.

For uncertainty, heteroscedastic approaches are very mature and have been used in similar tasks. One common approach predicts an input dependent variance and trains it through a Gaussian negative log likelihood loss function [6], however, this is known to converge poorly and instead, the β-NLL reweighting approach was designed to produce better results [17].

This study intends to produce dedicated per point uncertainty estimates with a voxelization process that is similar to other state of the art methods.

3 Method

3.1 Problem Formulation

LiDAR Scene Flow is done over two consecutive LiDAR sweeps. Let S_0 and S_1 denote the two consecutive LiDAR sweeps at times t_0 and t_1, respectively, where $\Delta t = t_1 - t_0 = 0.1$ seconds due to the sensor's 10Hz sampling rate [21]. Each sweep produces a point cloud P with points $p_i = (x_i, y_i, z_i, i_i) \in \mathbb{R}^4$, where $(x_i, y_i, z_i) \in \mathbb{R}^3$ and represent metric coordinates relative to the vehicle's LiDAR sensor.

Between consecutive sweeps, the vehicle, and thus its LiDAR sensor, moves in 3D space. The motion vector of the vehicle $M \in \mathbb{R}^3$ must be transformed between the two point clouds to make both LiDAR sweeps be in the same frame of reference. Thus, we apply $-M$ to the coordinate points in S_1 to change their coordinate reference frame to S_0's.

The objective of LiDAR Scene Flow is to predict where each individual point p moves from S_0 to S_1. Thus, the prediction must contain one flow vector $V \in \mathbb{R}^3$ for each point in S_0, showing where it has moved to relative to its initial starting point.

3.2 Data Preprocessing

Due to the Argoverse 2 Scene Flow challenge constraints, only points within a 140×140×6 meter box around the vehicle are considered: $x_i \in [-70, 70]$, $y_i \in [-70, 70]$, $z_i \in [-3, 3]$ [21]. Each point p_i also contains an int8 intensity scalar (0 - 255) representing the LiDAR's measured return strength. Points classified as ground by the dataset are removed from evaluation and points moving less than 0.5 m/s are also discarded.

Between S_0 and S_1, motion of the vehicle causes the coordinate frame of P_1 to be M away from P_0. By applying $-M$ onto every point $p_i \in P_1$, the coordinate frame of P_1 can be transformed to be identical to P_0, compensating for rigid egomotion of the vehicle. However, egomotion of the vehicle combined with the rolling shutter of a LiDAR sweep means that an object is not in its original position at the start and end of the LiDAR sweep [9] [24]. This is a known limitation [23] of only applying the vehicle's motion vector to P_1, and no attempt was made to compensate for it here. Figure 1 shows the voxelization process, an

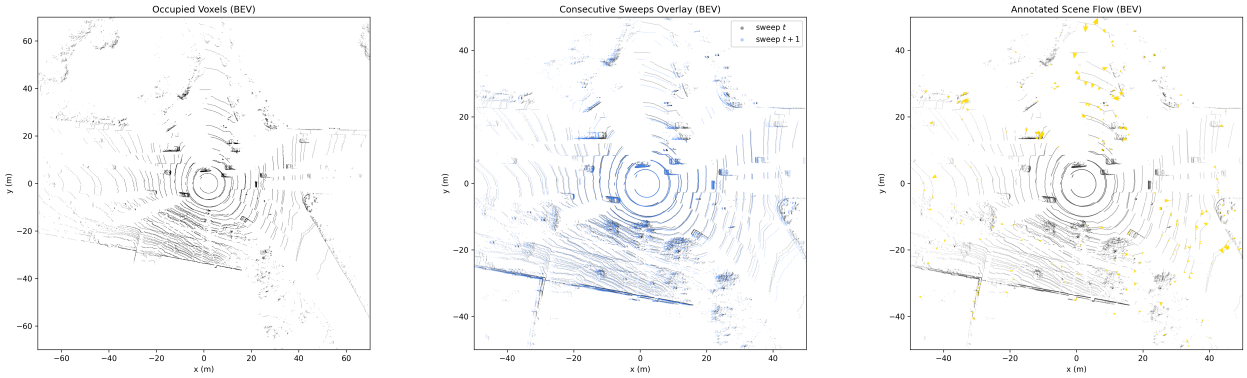

Figure 1: Voxelization and Data Annotation. The leftmost graph demonstrates sparse voxelization of a point cloud represented from a birds eye view. The center graph demonstrates the union of the LiDAR sweep and its following sweep, where the second sweep has had the ego motion of the vehicle subtracted as described in Section 3.2 to correct for the vehicle's motion. The rightmost graph has the union of the two LiDAR sweeps and the provided Argoverse annotations of the LiDAR Scene Flow.

overlay of both ego motion corrected sweeps, and the annotations provided for each voxel's flow.

The voxelizer converts each ego motion compensated point cloud into a sparse voxel grid. In each point cloud, cubic voxels are created with side lengths of 0.2m. Each voxel containing at least one point is then encoded into vector in $\mathbb{R}^4$ given as (x, y, z, i), while voxels with no poitns are discarded, hence sparse. The coordinates x, y, and z represent the average coordinate of all the points within the voxel relative to the center of the voxel. i represents the average intensity of each point in the voxel. In addition to all the voxel vectors, the inverse mapping for each voxel to each point is also retained, so during the devoxelization process, each voxel's flow vector can be mapped back to individual points within the voxel.

Using the voxelizer, both S_0 and S_1 are voxelized independently and then concatenated to create a single input vector in $\mathbb{R}^{10}$. The final vector is given as $(x_{0_i}, y_{0_i}, z_{0_i}, i_{0_i}, o_0, x_{1_i}, y_{1_i}, z_{1_i}, i_{1_i}, o_1)$. The coordinate points and intensity are derived from each individual sweep's voxelization process. The two additional parameters, o_0 and o_1 are either 0 or 1 to denote if the voxel was occupied in S_0 or S_1. For example, if in S_0, a particular voxel was unoccupied, $o_0 = 0$ and $o_1 = 1$. If a voxel is unoccupied in one of the sweeps, its x, y, z, and intensity all default to 0.

3.3 Architecture

The scene flow predictor is designed to be a minimum viable network that mirrors the basics of state of the art approaches such as DeltaFlow [23] and UniFlow [8]. It is a sparse three-dimensional UNET that takes the previously described 1x10 input vector and produces a 32 channel sparse tensor at the original resolution. Then, the flow is applied to individual points based on their parent voxel's flow vector.

The UNET uses three primary segments. First, there are submanifold segments which are two 3 x 3 x 3 submanifold convolutions which each then batch normalize and apply a ReLU nonlinearity. Second, there are the downsampling segments which use one 3 x 3 x 3 sparse convolutions with stride 2, which also then batch normalize and apply a ReLU. Third, there are upsampling blocks which use one inverse convolution, again followed by batch normalize and a ReLU.

The encoder starts with a submanifold, and then alternates between downsampling and more submanifolds, with three downsampling steps and four submanifold steps. The channel width thus grows from $10 \rightarrow 32 \rightarrow 64 \rightarrow 128 \rightarrow 256$. The decoder is a near inverse of the encoder, with alternating submanifolds and upsampling steps. The channel widths go from $256 \rightarrow 128 \rightarrow 64 \rightarrow 32$. The upsampled features are concatenated along the channel dimension with the equivalent encoder layer through a skip connection, for three skip connections. The architecture is visually shown in Figure 2.

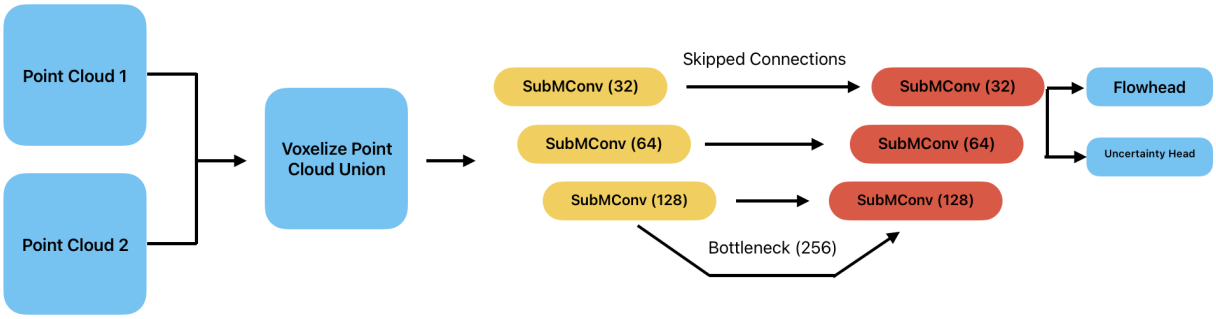

Figure 2: Data processing pipeline used by the system. First, the two point clouds are motion corrected and unionized. Then, the two point clouds are fed through the encoder decoder system and the final output is then outputted to both the flow head and the uncertainty head to produce the predicted flow and estimated uncertainty. Numbers in brackets represent channels, SubMConv is for SubManifold Convolution, and the yellow and red represent the encoder and decoder portions respectively.

The output of the encoder decoder is an individual voxel's 32 channel vector. This final output is parsed into two independents heads. The flow head produces flow per individual voxel, while the uncertainty head determines the confidence in the model's guess.

The flow head is a single transformation with a bias which can be defined as the flow vector $f_v = Th + b$ where h is the output of the decoder and the transform $T \in \mathbb{R}^{3\times32}$ and the bias $b \in \mathbb{R}^3$. For the sake of simplicity, every point in a voxel was assigned the flow vector of its parent voxel. However, there is much literature on different and superior methods such as interpolation [15], neural networks [5], attention [19], and rigidity [3].

The uncertainty head is produced by a simple multi-layer perceptron also taking in the decoder's output. It operates on the 1x32 feature vector and applies first a linear layer, ReLU, and then another linear layer which goes to a single value, which represents $\log \sigma^2$, or the log of the variance [6].

3.4 Training

The training of the encoder decoder and the flow transformation was done with endpoint error (EPE) defined as: $\mathcal{L}_{\text{EPE}} = \frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}} \left\|\hat{f}_i - f_i\right\|_2$. Where $\mathcal{L}_{\text{EPE}}$ is the EPE loss, $\mathcal{V}$ is the set of valid points, $\hat{f}_i \in \mathbb{R}^3$ is the predicted flow vector for a point i, and $f_i \in \mathbb{R}^3$ is the ground truth flow vector for a point i.

The uncertainty head is supervised by beta negative log likelihood (β-NLL) loss where the per point negative log likelihood can be defined as:

$$\mathcal{L}_{\beta\text{-NLL}} = \frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}} g(\sigma_i^2)^\beta \left(\frac{\|\hat{f}_i - f_i\|_2^2}{2\sigma_i^2} + \frac{3}{2}\log\sigma_i^2\right)$$

. Where $\mathcal{L}_{\beta\text{-NLL}}$ is the β-NLL loss, $g(\sigma_i^2)$ is the weighting function of the predicted variance [17], and β is the beta reweighting hyperparameter [13].

Additionally, at the end of training, a scalar s was applied as calibration to minimize the mean gaussian negative log-likelihood with respect to s:

$$\mathcal{L}_{\text{NLL}}(s) = \frac{1}{N}\sum_{i\in\mathcal{V}}\left[\frac{\|\hat{f}_i - f_i\|^2}{2s^2\sigma_i^2} + \frac{3}{2}\log s^2\right]$$

[6]. Where $\mathcal{V}$ is the set of valid points, $N = |\mathcal{V}|$, $\hat{f}_i \in \mathbb{R}^3$ is the predicted flow vector for point i, $f_i \in \mathbb{R}^3$ is the corresponding ground truth flow vector, and σ_i^2 is the predicted variance output by the uncertainty head before scaling [13]. The factor $\frac{3}{2}$ arises from the log-determinant of the isotropic covariance [17]. So:

$$s = \sqrt{\frac{1}{3N}\sum_{i\in\mathcal{V}}\frac{\|\hat{f}_i - f_i\|^2}{\sigma_i^2}}$$

During training, two separate experiments were conducted. In both, the encoder decoder and the flow head were both trained first. Then in Experiment A, the encoder decoder and flow head were frozen, and only the uncertainty head was trained. Then in Experiment B, the encoder decoder was frozen, and both the flow head and the uncertainty head were trained together.

4 Experiments

4.1 Environment Setup and Datasets

All training and experimentation was done on the officially provided Argoverse 2 Sensor dataset. Training was done on a single Nvidia Tesla V100 GPU on the NRP Nautilus platform. The training run for the encoder decoder and flow head previously described took about 10 days of continuous training with 25 epochs until convergence, while the Experiments A and B both took an addition 3 - 4 hours of training each. More training details are provided in the appendix. As previously described, in both experiments, the encoder decoder weights remained frozen while the uncertainty head was trained with β-NLL supervision. However, in Experiment A, the flow prediction head was frozen, and in Experiment B, the flow prediction head was trained jointly with the uncertainty prediction head.

The Argoverse dataset consists of 700 training scenes and 150 validation scenes lasting around 15 seconds, each broken into individual logs in Apache Feather formats [21]. This totals to almost 110,000 separate sweeps, or about 55,000 separate scenarios for training. All results in this section are reported on the validation set split, as test set annotations are not provided for the challenge. The sensor suite involves LiDAR sweeps at 10Hz coming from two stacked 32 beam sensors not corrected for egomotion. Additionally, there are 7 ring cameras and 2 stereo cameras with 20fps imagery. The camera systems were not used in any part of the experiments.

4.2 Evaluation Metrics

Five metrics were used to assess the accuracy of the uncertainty head: mean predicted σ, root mean squared error, gaussian negative log likelihood, expected normalized calibration error, and coverage.

σ denotes the predicted standard deviation of the 3D gaussian distribution over the flow vector in all three dimensions. The mean predicted σ describes the predicted size of the confidence region for a specific flow prediction. Root mean squared error (RMSE) describes the endpoint error in meters and describes the actual prediction error occurred from the ground truth labels. The gaussian negative log likelihood (NLL) scores the model on its ability to predict uncertainty with the accuracy of the flow prediction and the predicted uncertainty. Expected normalized calibration error (ENCE) [14] is computed by binning points into equally sized bins and predicting the average σ and averaging the relative discrepancy between binned RMSE and mean σ. Coverage at a confidence level α is the fraction of points whose flow error falls within the predicted α-level confidence region.

4.3 Results

Both Experiments A and B converged quickly within three epochs of training on the large dataset but both experienced variance collapse. Variance collapse is a failure mode in heteroscedastic uncertainty estimation where it learns to output a very low variance regardless of input which produces a locally stable solution under gaussian NLL training, despite the result being quite poor [2] [17]. The middle 50 percent of all predictions fell within 0.0006m for Experiment A and 0.0010 for Experiment B, which both predicted less than 70% of error at a 90% confidence interval as shown in Table 1. While the joint training in Experiment B had a larger window, the variance still collapsed significantly and a large portion of flow predictions do not fall within the uncertainty range, making both models significantly overconfident.

Using a scalar to calibrate variance networks such as both of these uncertainty heads that experience variance collapse is a known method to improve results [7]. Adding the previously defined scalar to the predicted uncertainty, better results can be achieved without altering the flow head as shown in Table 1. A single scalar corrects the ENCE down 32x and 54x to 0.042 and 0.025 for Experiments A and B respectively, with the 90% confidence interval achieving nearly 90% coverage. While Experiment B prior to scaling has better NLL results than Experiment A, after scaling both methods achieve very comparable coverage, ENCE, and NLL values.

The scaling corrects the global error results, but cannot correct the class differentiation ability of both experiments. Table 2 shows that Experiments A and B exhibit little variation in predicted σ across classes despite significant differences in prediction difficulty.

However, the predicted error variance is minimal, within 0.001 - 0.005 meters between Experiments A and B. This indicates the models learned the average dataset difficulty but failed to discriminate between

Experiment	ENCE	NLL	Coverage 90%	Coverage 95%
A	1.420	2.16	0.658	0.761
A + Scale	0.042	0.65	0.88	0.94
B	1.362	2.05	0.674	0.779
B + Scale	0.025	0.65	0.89	0.94

Table 1: Results made by both Experiments A and B, both scaled and unscaled. The unscaled results had variance collapse, but the scaled factor is able to correct for it, and produce significantly lower expected normalized calibration error (ENCE) and lower loss by negative log likelihood (NLL).

Bucket	n	σ (A + Scale)	σ (B + Scale)	RMSE (A + Scale)	RMSE (B + Scale)
Bicyclist	4,675	0.193	0.198	0.477	0.479
Motorcyclist	2,505	0.194	0.199	0.414	0.416
Wheeled-rider	4,864	0.196	0.202	0.455	0.457
Other Foreground	677,030	0.193	0.194	0.485	0.488
Background	17,027,147	0.193	0.197	0.466	0.466

Table 2: Bucket classification results between scaled experiments A and B. Both models failed to distinguish the difficulty of different objects in point clouds, such as bicyclists, and produce very similar accuracy between class as well.

classes. Interestingly, motorcyclists have the lowest RMSE despite being the smallest sample size and among the most difficult to predict due to their motion.

4.4 Discussion

Across all the calibration metrics in Table 1, Experiment B does negligibly better than Experiment A with lower ENCE (1.362 vs 1.420) and the scaled results also provide better results (0.025 vs 0.042). The coverage at the 90% and 95% confidence intervals was slightly better prior to scaling in Experiment B again, with about a 2% advantage in both. However, after scaling, both Experiments A and B have near identical results in coverage. Despite Experiment B's slightly better results, Table 2 shows that across all categories, Experiment B consistently has a very slightly larger σ, creating a larger uncertainty region. Additionally, the RMSE in Table 2 shows that Experiment B with its retrained flow prediction head predicts slightly inferior results compared to Experiment A. However, the RMSE across all categories is always within 3 thousandths between Experiments A and B, making the difference in accuracy negligible.

Experiment A and B have negligible differences, and the two are roughly tied in the result. Both results produced nearly identical results between RMSE (within 3 thousandths) and near identical coverage (within 1%) at both the 90% and 95% confidence intervals. However, because both experiments collapsed identically, it is likely that the frozen encoder decoder does not carry sufficient information to properly train the model. Fine tuning, perhaps with a lower learning rate over the encoder decoder, could produce better results and the decoder output may contain more important information to the uncertainty head.

5 Conclusion

This work proposes a lightweight add on for scene flow estimation to predict uncertainty. Both experiments show that a lightweight uncertainty head can be attached to a pretrained backbone at a low compute cost, but variance will collapse to a near constant value. Scaling the values also proved very effective and was able to reduce ENCE significantly and bring the 95% uncertainty coverage to 94% in both experiments. These scaled values are still useful, and the model can provide general uncertainty estimates for the LiDAR at a given moment despite not being able to accurately provide per object nor per class estimates.

However, a major limitation is that the uncertainty flow heads were both unable to assign meaningfully different uncertainty between classes, and were unable to understand that moving objects, like bicyclists, are

typically harder to predict than stationary objects in the background. This likely suggests that the decoder output does not contain information to discriminate classes, and could stem from the simplicity of the model, such as the very simple ego motion correction, and lack of any rolling shutter correction. Additionally, the flow prediction method may just not have been powerful enough, as the RMSE for bicyclists is the same as the background, showing the model was not struggling with them more than other objects.

Future work should investigate adding uncertainty flow prediction heads on preexisting state of the art models such as DeltaFlow or UniFlow and determining how well the uncertainty head can assess uncertainty in them. Additionally, unfreezing the encoder decoder may allow the decoder output to contain more features that are relevant to predicting uncertainty and produce more accurate predictions.

Throughout the process of this thesis, I learned about conducting large training runs from scratch. Prior to this thesis, I had never trained such a large neural network (almost 10 million parameters) from scratch before. It is also by far the largest dataset I have ever worked with. One of the biggest challenges I encountered was the loss functions kept ending up getting weird exceptions and either dropping to zero, infinity, or NaN. Ultimately, I had to add code to catch potential loss exceptions and irregularities to ensure training would continue.

References

[1] Cesar Cadena et al. "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6 (Dec. 2016), pp. 1309–1332. ISSN: 1941-0468. DOI: 10.1109/tro.2016.2624754. URL: http://dx.doi.org/10.1109/TRO.2016.2624754.

[2] Nicki S. Detlefsen, Martin Jørgensen, and Søren Hauberg. *Reliable training and estimation of variance networks*. 2019. arXiv: 1906.03260 [stat.ML]. URL: https://arxiv.org/abs/1906.03260.

[3] Ayush Dewan et al. "Rigid scene flow for 3D LiDAR scans". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 1765–1770. DOI: 10.1109/IROS.2016.7759282.

[4] Jingyun Fu et al. "PT-FlowNet: Scene Flow Estimation on Point Clouds With Point Transformer". In: *IEEE Robotics and Automation Letters* 8.5 (2023), pp. 2566–2573. DOI: 10.1109/LRA.2023.3254431.

[5] Philipp Jund et al. "Scalable Scene Flow from Point Clouds in the Real World". In: (2021). arXiv: 2103.01306 [cs.CV]. URL: https://arxiv.org/abs/2103.01306.

[6] Alex Kendall and Yarin Gal. *What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?* 2017. arXiv: 1703.04977 [cs.CV]. URL: https://arxiv.org/abs/1703.04977.

[7] Dan Levi et al. "Evaluating and Calibrating Uncertainty Prediction in Regression Tasks". In: *CoRR* abs/1905.11659 (2019). arXiv: 1905.11659. URL: http://arxiv.org/abs/1905.11659.

[8] Siyi Li et al. *UniFlow: Towards Zero-Shot LiDAR Scene Flow for Autonomous Vehicles via Cross-Domain Generalization*. 2025. arXiv: 2511.18254 [cs.CV]. URL: https://arxiv.org/abs/2511.18254.

[9] Yancong Lin and Holger Caesar. *ICP-Flow: LiDAR Scene Flow Estimation with ICP*. 2024. arXiv: 2402.17351 [cs.CV]. URL: https://arxiv.org/abs/2402.17351.

[10] Jiuming Liu et al. *DifFlow3D: Toward Robust Uncertainty-Aware Scene Flow Estimation with Diffusion Model*. 2024. arXiv: 2311.17456 [cs.CV]. URL: https://arxiv.org/abs/2311.17456.

[11] Xingyu Liu, Charles R. Qi, and Leonidas J. Guibas. *FlowNet3D: Learning Scene Flow in 3D Point Clouds*. 2019. arXiv: 1806.01411 [cs.CV]. URL: https://arxiv.org/abs/1806.01411.

[12] Yufei Liu et al. "Self-Supervised Diffusion-Based Scene Flow Estimation and Motion Segmentation With 4D Radar". In: *IEEE Robotics and Automation Letters* 10.6 (2025), pp. 5895–5902. DOI: 10.1109/LRA.2025.3563829.

[13] D.A. Nix and A.S. Weigend. "Estimating the mean and variance of the target probability distribution". In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*. Vol. 1. 1994, 55–60 vol.1. DOI: 10.1109/ICNN.1994.374138.

[14] Pascal Pernot. *Properties of the ENCE and other MAD-based calibration metrics*. 2023. arXiv: 2305.11905 [cs.LG]. URL: https://arxiv.org/abs/2305.11905.

[15] Charles R. Qi et al. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*. 2017. arXiv: 1706.02413 [cs.CV]. URL: https://arxiv.org/abs/1706.02413.

[16] Samuel Schulter et al. *Deep Network Flow for Multi-Object Tracking*. 2017. arXiv: 1706.08482 [cs.CV]. URL: https://arxiv.org/abs/1706.08482.

[17] Maximilian Seitzer et al. "On the Pitfalls of Heteroscedastic Uncertainty Estimation with Probabilistic Neural Networks". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=aPOpXlnV1T.

[18] Deqing Sun et al. *PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume*. 2018. arXiv: 1709.02371 [cs.CV]. URL: https://arxiv.org/abs/1709.02371.

[19] Guangming Wang et al. "Hierarchical Attention Learning of Scene Flow in 3D Point Clouds". In: *IEEE Transactions on Image Processing* 30 (2021), pp. 5168–5181. ISSN: 1941-0042. DOI: 10.1109/tip.2021.3079796. URL: http://dx.doi.org/10.1109/TIP.2021.3079796.

[20] Anne S. Wannenwetsch, Margret Keuper, and Stefan Roth. "ProbFlow: Joint Optical Flow and Uncertainty Estimation". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 1182–1191. DOI: 10.1109/ICCV.2017.133.

[21] Benjamin Wilson et al. "Argoverse 2: Next Generation Datasets for Self-Driving Perception and Forecasting". In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks 2021)*. 2021.

[22] Ekim Yurtsever et al. *3D Object Detection with a Self-supervised Lidar Scene Flow Backbone*. 2022. arXiv: 2205.00705 [cs.CV]. URL: https://arxiv.org/abs/2205.00705.

[23] Qingwen Zhang et al. "ΔFlow: An Efficient Multi-frame Scene Flow Estimation Method". In: *NeurIPS* (2025).

[24] Qingwen Zhang et al. *himo: high-speed Objects Motion Compensation in Point Clouds*. 2025. DOI: 10.1109/TRO.2025.3619042. arXiv: 2503.00803 [cs.CV]. URL: https://arxiv.org/abs/2503.00803.

[25] Yushan Zhang et al. *DiffSF: Diffusion Models for Scene Flow Estimation*. 2024. arXiv: 2403.05327 [cs.CV]. URL: https://arxiv.org/abs/2403.05327.

[26] Sifan Zhou et al. *FastPillars: A Deployment-friendly Pillar-based 3D Detector*. 2023. arXiv: 2302.02367 [cs.CV]. URL: https://arxiv.org/abs/2302.02367.

6 Appendix

6.1 Training Details

Training was done in a Kubernetes pod with a single Nvidia Tesla V100 SXM2 16GB GPU. Python packaged by Anaconda 3.10.9, CUDA 11.8, PyTorch 2.7.1, SPConv 2.3.4 were the major libraries used during training. Mixed precision training was done using FP16 for the encoder decoder and FP32 for the β-NLL loss.

Due to limitations in the read speed from the storage used (Rook CephFS with ReadWriteMany for a Persistent Volume Claim on NRP Nautilus) the entire egomotion, voxelization, and encoding process was done a single time and all the data was serialized with PyTorch and stored for faster read and writes.

Weights were initialized using PyTorch's default Kaiming uniform initialization. The AdamW optimizer was used.

6.2 Codebase

The codebase is open source and MIT licensed here: https://github.com/DanielRhee/Lidar-Flow