# Data Scaling Laws for End-to-End Autonomous Driving

## Supplementary Material

## A. More Details on the Dataset

We analyze the distribution of the vehicle's speed and the distance to take actions (i.e. lane changing and taking a turn). We group these statistics into bins with an interval of 10 km/h for the speed and 2 m for the action distance. The distributions are provided in Fig. 7 and Fig. 8 respectively.

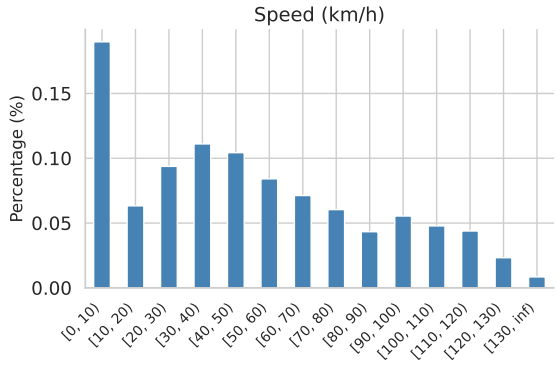


Figure 7. Speed distribution of the ego vehicle.

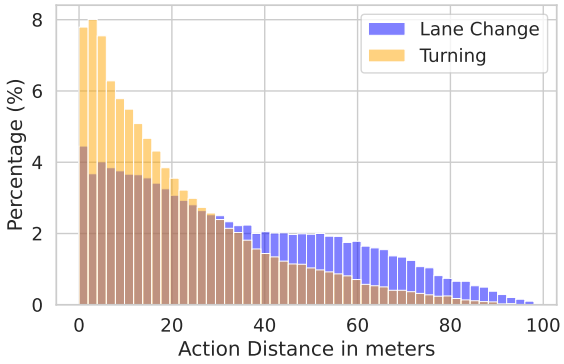


Figure 8. Distribution of the action distance.

### A.1. Data Quality Analysis

Our data pre-processing pipeline relies on auto-labeling, which may result in some portion of noisy samples i.e. with inaccurate turn angles. We validated label quality by comparing turn angles (local linear approximations of lanelets) with their global counterpart (linear approximations of full lanelets). Fig. 9 (left) shows where they match (green) and where they don't (red) and Fig. 9 (right) shows the ratio of accurately labeled turn angles for different angle bins and we show that the accuracy of our labeling drops in sharper turns.

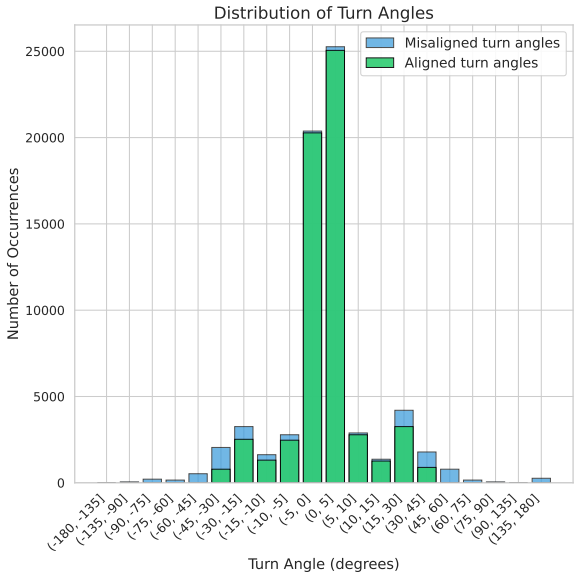


Figure 9. Turn angle alignment.

### A.2. Data Curation Procedure

We provide some additional information for the dataset curation pipeline in the following.

**Geofencing.** We utilize an H3 cell resolution of 11, corresponding to a cell edge length of 24.91m and a cell diameter of 49.82m.

**ODD Distribution.** Maintaining near-identical ODD proportions in splits smaller than 32 hours remains challenging, especially when factoring in large session clusters. We investigated breaking up these clusters by analyzing which H3 cells are responsible for large connected regions and assessing what would happen if they were removed, but ultimately opted against this strategy because the cluster connections were too complex and dense in our data. Instead, to obtain the final splits, we iteratively grow each dataset portion by randomly sampling a session cluster from the smallest 50% of the remaining ones and assigning it to the split that best preserves the desired distribution. Repeating this approach multiple times and selecting the configuration closest to the real-world distribution yielded the best results, and we verified that this ODD distribution remains valid across all dataset sizes.

**Action Labeling.** Due to issues with the temporal consistency of the localization when using lanelet matching

[55] based on position and orientation, we implemented a *trajdata* extension that generates diverse map-based anchor paths (DMAPs) following [52]. This enables the computation of map-based anchor paths for the ego-vehicle from any initial position. For each anchor path, we measure alignment with the ground truth ego-motion to determine the ego's future lane sequence. The matching score is computed as

$$s = \alpha s_{\text{IoU}} + (1 - \alpha) s_{\text{LI}}$$

where $s_{\text{IoU}}$ is the Intersection over Union (IoU) of the ego and anchor paths (each with 1m of buffer), $s_{\text{LI}}$ is the percentage of the ego path within the lanelets given by the anchor path, and weighting factor $\alpha \in [0, 1]$. We save the action distance in a global reference frame, i.e. as driven distance from the session start. Since multiple snapshots will contain the same action, this global reference frame enables the removal of noisy snapshot data by using majority voting to determine the final action. To simplify the action encoding for training, we only take actions within the ground truth traveled distance into account and save only one action conditioning input per snapshot, although multiple inputs might have been generated. We use manual visual debugging to verify the procedure qualitatively.

### A.3. Corner Case Handling

We focus on analyzing overall model performance improvement as dataset size increases and provide a more comprehensive understanding of AV data-scaling laws, rather than analyzing corner cases. Detecting corner cases would require (subjective) definitions, targeted labeling, and additional mechanisms like novelty or out-of-distribution detection, which are out of the scope of this work.

## B. Model Diagram and Details

We provide details on our perception module in Appendix B.1.

### B.1. Details on the Perception Module

We provide further clarification on the variables and operations introduced in the Perception Module. The perception module processes input images from multiple camera views and extracts feature representations for downstream tasks. Below is a brief explanation of the key variables used:

- $\mathbf{I}_v \in \mathbb{R}^{H \times W \times 3}$: The rectified image from each camera view, where $v$ denotes the camera view (front $f$, left $l$, or right $r$).
- $\mathcal{E}(\cdot)$: The ResNet-based encoder used to extract features from each input image. It includes a global average pooling layer that reduces the spatial dimensions of the feature maps to a single vector.
- $\mathbf{F}_v = \mathcal{E}(\mathbf{I}_v) \in \mathbb{R}^d$: The encoded feature vector for each view $v$, where $d$ is the dimension of the feature vector after global pooling.

The multi-view fusion process leverages cross-attention to combine information from all available views (front, left, and right), ensuring balanced integration of lateral perspectives:

1. Define the front view's feature map, $\mathbf{F}_f$, as the query ($\mathbf{Q}$), and use the left and right feature maps, $\mathbf{F}_l$ and $\mathbf{F}_r$, as keys ($\mathbf{K}$) and values ($\mathbf{V}$) in separate cross-attention layers.
2. Apply cross-attention to combine features, where each cross-attention layer updates $\mathbf{F}_f$ by attending to $\mathbf{F}_l$ and $\mathbf{F}_r$:

$$\mathbf{A}_{f,l} = \text{CrossAttn}(\mathbf{F}_f, \mathbf{F}_l)$$
$$\mathbf{A}_{f,r} = \text{CrossAttn}(\mathbf{F}_f, \mathbf{F}_r)$$

   Here, $\mathbf{A}_{f,l}$ and $\mathbf{A}_{f,r}$ represent the attention outputs for the front-left and front-right interactions, respectively. These outputs are then aggregated to form the final fused representation.
3. **Symmetric Fusion**: The final fused feature map, $\mathbf{F}_{\text{fused}}$, is computed by aggregating the cross-attention outputs. We use a simple element-wise summation:

$$\mathbf{F}_{\text{img}} = \mathbf{F}_f + \mathbf{A}_{f,l} + \mathbf{A}_{f,r}$$

   This symmetric fusion captures contextual information from both lateral views equally, enhancing the spatial awareness of the front view.

## C. Impact of Varying Training Points on Scaling Law Estimators

As illustrated in Fig. 11 and Tab. 4, incorporating more data points to train the **M2** estimator on FDE values enables a closer fit to the scaling curve, improving the alignment with the data and leading to lower extrapolation loss on the test set.

## D. Selecting Scaling Law Estimators

As described in the method section, the estimators are trained using the first six data points, with the 1024-hour and 2048-hour points reserved for validation. Figure 10 illustrates the scaling law fitting on FDE across all scenarios. Among the estimators, **M2** and **M4** demonstrate the best fit, achieving the lowest mean squared error (MSE). In contrast, **M1** fails to capture the trend and reduces to a straight line (a pure power law), while **M3** provides overly optimistic estimates, with values decreasing too quickly toward the end, resulting in a higher MSE. Following Occam's Razor, we choose **M2** over **M4** in our analysis.

Moreover, we believe the discrepancies in scaling law curve-fitting arise more from the inherent challenges of scaling law estimation than dataset quality, as existing estimators exhibit varying levels of robustness across different

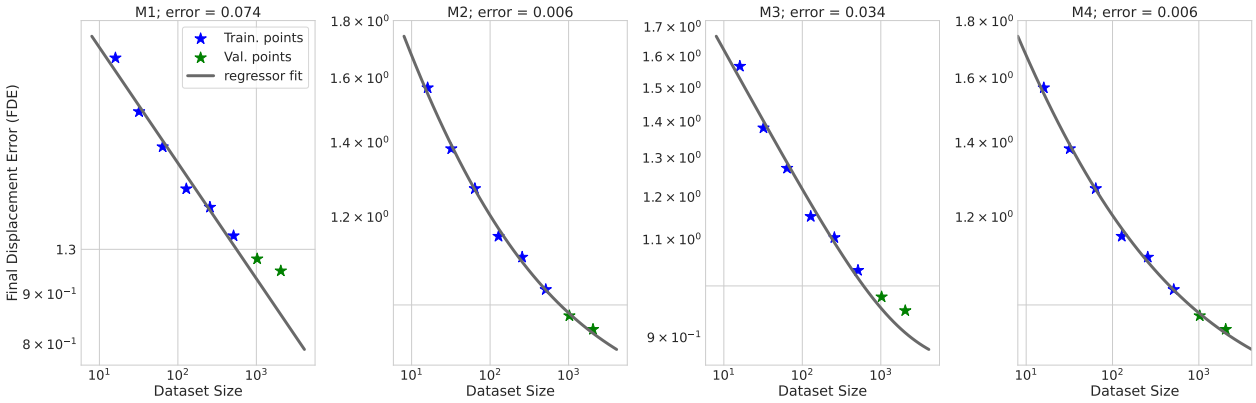


Figure 10. Comparison of fitting all scaling law estimators on the full dataset.

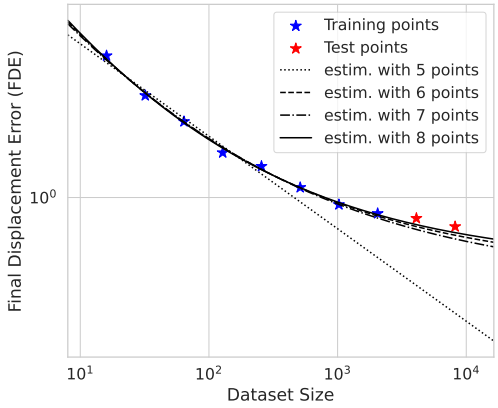


Figure 11. Analyzing the performance for **M2**: $y - \epsilon_\infty = \beta x^c$ in the case of iteratively increasing the number of training points.

| # Points | $\beta$ | $c$ | $\epsilon_\infty$ | Extrapolation Loss |
|---|---|---|---|---|
| 5 | 2.1837 | -0.1274 | 0.0000 | $0.2002 \pm 0.0135$ |
| 6 | 1.9010 | -0.3319 | 0.7917 | $0.0244 \pm 0.0002$ |
| 7 | 1.8594 | -0.3133 | 0.7663 | $0.0338 \pm 0.0009$ |
| 8 | 1.9488 | -0.3486 | 0.8103 | $0.0179 \pm 0.0004$ |

Table 4. Quantitative extrapolation results for **M2**: $y - \epsilon_\infty = \beta x^c$ using an iteratively increasing numbers of training points, complementing the visualization in Fig. 11.

data regimes. Collecting measurements averaged over multiple runs could reduce noise and improve curve-fitting, but would require significantly more compute power and time. A piecewise function could reduce error, but would introduce artificial discontinuities. Additionally, the plateau becomes apparent only for datasets exceeding $10^3$ hours, with metrics like FDE and ADE improving steadily in a near-linear trend on a log-log scale before this point. This reflects natural diminishing returns at larger scales, also seen in scaling law analyses in the computer vision and natural language domains, rather than a limitation of data quality.

# E. Experimental Setup

## E.1. Hyperparameters

We use the Adam optimizer for training our models without applying any weight decay. Training is conducted in FP32 precision, as we encountered instabilities when using FP16 or BF16 precisions. We employ a cosine annealing schedule for the learning rate, with the final learning rate set to 0 (i.e., $\eta_{min} = 0$). The initial learning rate ($\eta_{max}$) is scaled linearly with the total effective batch size in distributed training. Specifically, we use a learning rate of $0.001$ for an effective batch size ($bs$) of $1024$, and scale the initial learning rate for other batch sizes accordingly:

- $bs = 512 \rightarrow \eta_{max} = 0.0005$
- $bs = 1024 \rightarrow \eta_{max} = 0.001$
- $bs = 2048 \rightarrow \eta_{max} = 0.002$
- etc.

## E.2. Training Time and Hardware

We use a compute cluster consisting of A-100 GPUs. Particularly, training a model with the ResNet-18 backbone on the largest data split (8192 hours) takes around 24 hours on 8×A-100 GPUs.