When Anchors Meet Cold Diffusion: A Multi-Stage Approach to Lane Detection

Supplementary Material

```
Algorithm 1: Training algorithm
  Require: Restoration model $R_\theta$, lane degradation process $D_\alpha$, image degradation process $D_\gamma$
  Input: Image $\mathbf{I}$, predefined anchors $\mathbf{A}$
  Output: Training loss
1 Convert image $\mathbf{I}$ to resolution $\gamma_T$, $\mathbf{I}_T \leftarrow D_\gamma(\mathbf{I}, T)$;
2 Obtain initial prediction $\hat{\mathbf{A}}^0 \leftarrow R_\theta(\mathbf{A}, T, \mathbf{I}_T)$;
3 Choose a random timestep $t \leftarrow$Uniform$(\{1, \ldots, T\})$;
4 Convert image $\mathbf{I}$ to resolution $\gamma_t$, $\mathbf{I}_t \leftarrow D_\gamma(\mathbf{I}, t)$;
5 Produce degraded lanes $\mathbf{A}^t \leftarrow D_\alpha(\hat{\mathbf{A}}^0, t)$; (using Eq. 6)
6 Predict result by $R_\theta(\mathbf{A}^t, t, \mathbf{I}_t)$;
7 Calculate the training loss $\mathcal{L}_{total}$; (using Eq. 9)
```

```
Algorithm 2: Inference algorithm
  Require: Restoration model $R_\theta$, lane degradation process $D_\alpha$, image degradation process $D_\gamma$, total diffusion step $T$, interval $\Delta t$
  Input: Image $I$, predefined anchors $\mathbf{A}$
  Output: Final prediction result $\mathbf{A}^0$
1  Start from predefined lane anchor $\mathbf{A}$
2  for $t = T, T - \Delta t, \ldots, 1$ do
3      Convert image $\mathbf{I}$ to resolution $\gamma_t$, $\mathbf{I}_t \leftarrow D_\gamma(I, t)$;
4      Obtain prediction $\hat{\mathbf{A}}^0 \leftarrow R_\theta(\mathbf{A}, t, \mathbf{I}_t)$;
5      if $t > 1$ then
6          Refine target estimation via Eq. 4
           $\mathbf{A} \leftarrow \mathbf{A} - D_\alpha(\hat{\mathbf{A}}^0, t) + D_\alpha(\hat{\mathbf{A}}^0, t - \Delta t)$;
7      else
8          return $\hat{\mathbf{A}}^0$
9      end
10 end
```

A. Implementation Details

Architecture. Our restoration network R_θ builds on CLRNet [53]. It adopts either a ResNet [13] or DLA [50] backbone, followed by an FPN [21] to extract multi-scale feature maps. These feature maps are then cropped according to the degraded lane $\mathbf{A}^t$, and the resulting cropped features pass through several convolutional and fully connected layers to produce final lane predictions together with their associated confidence scores. Notably, this architecture modification affects only the input and the training objective, keeping the core pipeline structure intact. As a result, *CDiffLane* can be integrated into any anchor-based lane detection framework with minimal additional overhead.

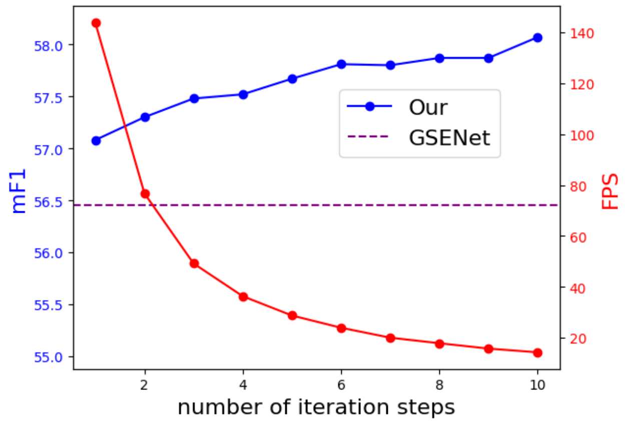

Figure 4. mF_1 and FPS with a different number of iteration steps.

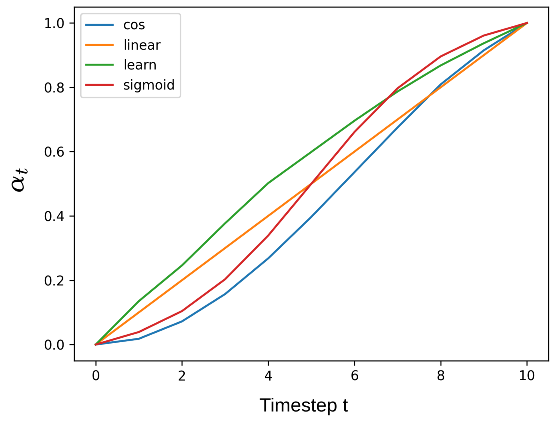

Figure 5. Comparisons with Different α.

Training Configuration. All input images are resized to 320×800 in both training and testing. We follow the data preprocessing strategies in [53], applying random translation, rotation, scaling, and horizontal flipping to enhance generalization. Our optimization uses the **AdamW** optimizer with an initial learning rate of 5×10^{-4} under a **cosine decay** schedule. On **CULane**, training proceeds for 15 epochs, while on **TuSimple**, it runs for 70 epochs; both use a batch size of 24.To stabilize the training process, we fix the learnable parameter α at the initial stages and allow it to become learnable after the model acquires sufficient knowledge about lane detection. All experiments are conducted in **PyTorch** on a single **NVIDIA RTX 4070** GPU. The number of sampling points is $N_s = 72$, the number of lane anchors is $N = 192$, and we adopt a linear schedule for γ, ranging from 1.8 to 1. For the loss weights, we set $\lambda_{\text{cls}} = 2$, $\lambda_{xytl} = 0.2$, and $\lambda_{iou} = 4$.

Inference Procedure. During inference, the network outputs a 6D vector $(x_p, y_p, \theta_p, l, \delta_x, c)$ per lane hypothesis, where c denotes the confidence score. We then apply **non-maximum suppression (NMS)** to filter out redundant detections, retaining only the highest-confidence lanes. This ensures the final lane set is both *accurate* and *minimally redundant*.

B. Evaluation Metrics

For the CULane dataset, we evaluate lane prediction accuracy using the F1-measure, computed via the Intersection-over-Union (IoU) between predicted and ground-truth lanes. A prediction is deemed a true positive (TP) if its IoU exceeds a given threshold; otherwise, it counts as a false positive (FP) or false negative (FN). Formally, the F1-measure is:

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall}, \tag{10}$$

where $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$. In our experiments, we report F_1@50 and F_1@75, which apply IoU thresholds of 0.5 and 0.75, respectively. Notably, the higher threshold (0.75) offers a more stringent test of precise lane localization, allowing us to highlight improvements in fine-grained accuracy. In addition, we also report mF_1 score as one of the metrics. It is defined as:

$$mF_1 = (F_1@50 + F_1@55 + \cdots + F_1@95)/10. \tag{11}$$

For the TuSimple dataset, a lane is deemed correct if more than 85% of its predicted points fall within a 20-pixel radius of the corresponding ground truth. Based on this criterion, the benchmark accuracy metric is computed as:

$$Accuracy = \frac{\Sigma_{clip} C_{clip}}{\Sigma_{clip} S_{clip}}, \tag{12}$$

where C_{clip} is the number of correctly predicted points and S_{clip} is the total number of ground-truth points of a image respectively. TuSimple tracks False Positive (FP) and False Negative (FN) rates, where $FP = \frac{F_{pred}}{N_{pred}}$ and $FN = \frac{M_{pred}}{N_{gt}}$.

C. Supplementary Experiments

Iterative evaluation. We perform an experiment to examine how mF_1 varies with the number of inference steps and the resulting speed (FPS). As shown in Fig. 4, the iterative strategy indeed improves performance but lowers the number of tasks the model can process per second. While this trade-off may be acceptable for tasks such as high-density map construction, it can be less ideal in time-critical settings. Notably, even when reducing the iterative steps to just one—yielding an FPS above 140—our CDiffLane still surpasses GSENet, highlighting its robustness under lower iteration budgets.

Learned α comparison. To evaluate the effectiveness of our approach, we compare the curve of the learned α with several commonly used variation schedules, including sigmoid, cosine, and linear. Initially α is set using a linear schedule and is continuously updated throughout the training process. The results, illustrated in Figure 5, indicate that the learned α follows a concave trajectory on a line chart, distinguishing it from all predefined schedules. This observation suggests that a fixed variation schedule may limit model performance, preventing optimal adaptation. Additionally, since we progressively provide images of increasing resolution during inference, it is reasonable that the learned α follows a concave pattern. At the initial timesteps, where low-resolution images contain minimal information, the model learns to reduce their contribution to the final prediction by lowering the variation magnitude. Conversely, at later stages, as high-resolution images become available, the model increases the variation magnitude, leveraging the enriched details to refine its predictions more effectively. This adaptive scheduling mechanism underscores the importance of a learnable variation schedule, enabling the model to achieve superior accuracy and robustness.