# SynAD: Enhancing Real-World End-to-End Autonomous Driving Models through Synthetic Data Integration

Supplementary Material

## A. Ego-centric Scenario Generation

### A.1. Diffusion Training.

For the backbone model, we leverage a U-Net backbone with custom temporal adaptations for handling sequential features and multi-agent scenarios. In the reverse process, the condition $\mathbf{f}$ is constructed by concatenating the map features from the past $h = 1$ timestamp. These map features are encoded using a ResNet model. We train the diffusion model on the nuScenes training set for 200 epochs using the Adam optimizer with a learning rate of $2 \times 10^{-4}$ and employ cosine learning rate decay.

### A.2. Guide Functions

**Agent Collision.** To prevent agents from colliding with each other, we introduce an agent collision guide function that penalizes trajectories where agents come too close. Each agent $i$ is approximated as a circle centered at its predicted position, with a radius $r_i$ defined as half the diagonal length. To maintain a safe separation between agents, we define a safety distance as the sum of their radius and a buffer distance $\delta$:

$$d_{\text{safe},ij} = r_i + r_j + \delta, \quad \text{where} \quad \delta = 1. \tag{A}$$

For each pair of agents $i$ and $j$, we denote the Euclidean distance between the two agents as $d^t_{ij}$. Then, we define the agent collision guide for agent $i$ at timestamp $t$ as:

$$R^t_{\text{agent},i} = \sum_{j \neq i} \max\left(1 - \frac{d^t_{ij}}{d_{\text{safe},ij}}, 0\right) \tag{B}$$

To emphasize avoiding collisions earlier in the trajectory, we apply an exponential decay weighting to the collision penalties computed across all timestamps $T$:

$$R_{\text{agent},i} = \sum_{t=1}^{T} w(t) R^t_{\text{agent},i}, \tag{C}$$

$$\text{where} \quad w(t) = \frac{\gamma^t}{\sum_{k=1}^{T} \gamma^k}, \quad \gamma = 0.9. \tag{D}$$

In practice, the collision guide is computed only for agents with non-zero velocity, $\mathbb{1}_{\text{agent}}(i) = 1$. This ensures that stationary agents are excluded from the collision penalty, and the final agent collision guidance is calculated overall $M$ agents as:

$$R_{\text{agent}} = \sum_{i=1}^{M} \mathbb{1}_{\text{agent}}(i) R_{\text{agent},i}. \tag{E}$$

**Map Collision.** We introduce a map collision guide function to ensure that agents remain in drivable areas and avoid off-road regions. The function provides gradients that guide the agent back onto the road by considering the spatial relationship between off-road and on-road points within the agent's bounding box. For each agent $i$ at timestamp $t$, we sample a set of points $P^t_i$ arranged in a $10 \times 10$ grid along the width and height within its bounding box. This set of points is then divided into an on-road set $O^t_i$ and an off-road set $F^t_i$ with as follows:

$$O^t_i = \{p \in P^t_i \mid \mathcal{M}(p) = 1\}, \tag{F}$$

$$F^t_i = \{p \in P^t_i \mid \mathcal{M}(p) = 0\}, \tag{G}$$

where $\mathcal{M}(p)$ returns 1 if the point $p$ is on-road and 0 otherwise. The map collision guide encourages off-road points $p_{\text{off}}$ to align more closely with the on-road region by minimizing their distance to the nearest on-road point $p_{\text{on}}$. Additionally, we apply the same exponential decay weighting function $w(t)$ as defined in the agent collision guide below:

$$R^t_{\text{map},i} = \sum_{p_{\text{off}} \in F^t_i} \left(1 - \min_{p_{\text{on}} \in O^t_i} \|p_{\text{on}} - p_{\text{off}}\|_2\right), \tag{H}$$

$$R_{\text{map},i} = \sum_{t=1}^{T} w(t) R^t_{\text{map},i}, \tag{I}$$

$\mathbb{1}_{\text{map}}(i)$ ensures the map loss is applied only to agents that are moving and partially on- and off-road(i.e., $O^t_i \neq \emptyset$ and $F^t_i \neq \emptyset$). We calculate the map collision guide as follows:

$$R_{\text{map}} = \sum_{i=1}^{M} \mathbb{1}_{\text{map}}(i) R_{\text{map},i}. \tag{J}$$

**Speed.** We employ a speed limit guide function to ensure that agents adhere to both a predefined maximum speed $v_{\text{max}}$ and a minimum speed $v_{\text{min}}$, promoting safe and controlled behavior. Let $v^t_i$ denote the speed for agent $i$ at timestamp $t$, and the speed limit guide is defined as the amount by which the predicted speed deviates from the allowable range $[v_{\text{min}}, v_{\text{max}}]$, computed as:

$$R^t_{\text{speed},i} = \max(v^t_i - v_{\text{max}}, 0) + \max(v_{\text{min}} - v^t_i, 0), \tag{K}$$

$$R_{\text{speed},i} = \sum_{t=1}^{T} w(t) R^t_{\text{speed},i}, \tag{L}$$

$$R_{\text{speed}} = \sum_{i=1}^{M} \mathbb{1}_{\text{speed}}(i) R_{\text{speed},i}, \tag{M}$$

where $w(t)$ is an exponential decay weighting function and $\mathbb{1}_{\text{speed}}(i)$ is an indicator function that evaluates to 1 for agents with non-zero velocity and 0 otherwise. Finally, the guide $\mathcal{J}$ is defined as follows:

$$\mathcal{J} = \sum_{i \in \{\text{agent, map, speed}\}} w_i R_i \tag{N}$$

In Table 7, the weights are set as $w_{\text{agent}} = 50, w_{\text{map}} = 1$, and $w_{\text{speed}} = 1$ when each respective guide is used. If a guide is not utilized, its corresponding weight is set to 0.

### A.3. Ego-centric Conversion

**Drawing Map.** To generate the input maps, we utilize the nuScenes map API to extract relevant map data. Focusing on an area of $60m \times 60m$ centered around the ego vehicle, we include the following components: [*'drivable area', 'road segment', 'lane', 'ped crossing', 'walkway'*]. We overlay other vehicles onto the map at their corresponding coordinates, accurately rendering each vehicle by incorporating their size and orientation.

**Rotation Matrix.** We now describe the rotation matrix used in Equation 8. As illustrated in Figure A, the new coordinate system for the ego agent is defined by placing the ego's position $(s_x, s_y)$ at the origin and aligning the vehicle's heading direction (north) with the $y$-axis. First, the position of an arbitrary point $(x, y)$ is translated to $(x - s_x, y - s_y)$ to account for the ego's position. Given that the vehicle's heading is rotated counterclockwise by $s_\theta$ radians relative to the original coordinate system, the rotation of the translated point is determined by rotating axis by $\frac{\pi}{2} - s_\theta$ radians clockwise about the origin. This is equivalent to counterclockwise rotation by the same radians. The standard rotation matrix for rotating a point counterclockwise by an angle $\alpha$ is:

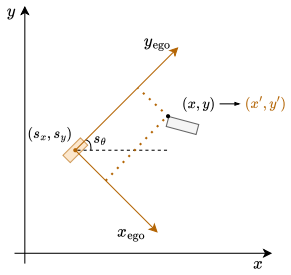

Figure A. The coordinate transformation for the ego-vehicle.

$$R(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{O}$$

Substituting $\alpha = \frac{\pi}{2} - s_\theta$ and applying trigonometric identities, the transformation is given by the following equation:

$$T(x, y; s) = \begin{pmatrix} \sin s_\theta & -\cos s_\theta \\ \cos s_\theta & \sin s_\theta \end{pmatrix} \times \begin{pmatrix} x - s_x \\ y - s_y \end{pmatrix}. \tag{P}$$

## B. E2E AD Network Details

### B.1. Motion Forecasting

Motion forecasting module predicts motion trajectories $\hat{\mathbf{x}} \in \mathbb{R}^{M \times N \times T \times 2}$ for $M$ agents over $T$ timestamps with possible $N$ series of waypoints $(x, y)$. We prepare $E_{\text{motion}}$, $E_{\text{agent}}$, and $E_{\text{ego}}$ by encoding normalized anchors and positional information through embedding layers with transformations. $E_{\text{motion}}$ contains information from anchors representing general motion patterns (e.g., turning left, going straight); $E_{\text{agent}}$ represents motion patterns in each agent's own coordinate system, focusing on motion offsets relative to the agent's current position and orientation; and $E_{\text{ego}}$ embeds how each agent's potential trajectory relates to the ego vehicle's position and orientation. By feeding these inputs to the MotionEncoder, which consists of Transformers and MLP layers, we generate the motion query embedding $q_{\text{motion}}$:

$$q_{\text{motion}} = \text{MotionEncoder}(E_{\text{motion}}, E_{\text{agent}}, E_{\text{ego}}). \tag{Q}$$

Using $q_{\text{motion}}$ and the BEV feature $B$, the MotionDecoder produces the refined motion query $\hat{q}_{\text{motion}}$,

$$\hat{q}_{\text{motion}} = \text{MotionDecoder}(q_{\text{motion}}, B). \tag{R}$$

The final motion prediction $\hat{\mathbf{x}}$ is then computed by feeding $\hat{q}_{\text{motion}}$ into MLP layers: $\hat{\mathbf{x}} = \text{MLP}(\hat{q}_{\text{motion}})$. Simultaneously, the model predicts the probabilities $p_k$ for each trajectory $\hat{\mathbf{x}}_k$ by passing $\hat{q}_{\text{motion}}$ through MLP layers, followed by a log softmax activation.

### B.2. Occupancy Prediction.

To estimate the future occupancy of the scene, we predict a sequence of occupancy maps $\hat{O} = \{\hat{O}^1, ..., \hat{O}^T\}$, where each $\hat{O}^t \in \mathbb{R}^{H \times W}$ corresponds to the occupancy at timestamp $t$. First, the instance-level embedding $q_{\text{ins}} \in \mathbb{R}^{M \times D}$ is derived from $\hat{q}_{\text{motion}}$ through MLP layers, where $M$ and $D$ are the number of agents and embedding dimension. At each timestamp $t$, we feed $q_{\text{ins}}$ into another MLP to generate temporal queries: $q^t_{\text{temp}} = \text{MLP}^t(q_{\text{ins}})$. Simultaneously, the raw BEV feature $B \in \mathbb{R}^{C \times H_{\text{bev}} \times W_{\text{bev}}}$ is reduced and downscaled to produce an initial state $B^0_{\text{state}} \in \mathbb{R}^{C \times \frac{H_{\text{bev}}}{4} \times \frac{W_{\text{bev}}}{4}}$. A transformer-based decoder, referred to as the OccDecoder, then updates $B^{t-1}_{\text{state}}$ by incorporating $q^t_{\text{temp}}$, producing an updated BEV feature $B^t_{\text{state}}$ as follows:

$$B^t_{\text{state}} = \text{OccDecoder}(B^{t-1}_{\text{state}}, q^t_{\text{temp}}). \tag{S}$$

After passing through additional upsampling, the final BEV features and the instance queries are fused via an element-wise dot product across the channel dimension, producing the occupancy logits $\hat{O}$.

# C. Training Details

## C.1. Map-to-BEV Network

We first initialize the BEV Query $Q_B$ of Map-to-BEV network from the pre-trained BEVFormer [17] We employ ResNet50 as the map encoder, utilizing only the layers up to the point before the pooling layer. Transformer encoder comprises 6 blocks, each including a cross-attention layer, a feedforward network, and normalization layers with residual connections. Our training setup involves the AdamW optimizer with a cosine annealing scheduler over 20 epochs, including a warm-up phase during the first 5 epochs. We set the learning rate to $5 \times 10^{-4}$ and apply a weight decay of 0.01. The momentum parameters are configured with $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

In Table 6, we implement the SwinUNETR architecture following the official implementation. When utilizing SwinUNETR, we set the feature size to 48 and the drop rate to 0.2 and increase the input map size to 800 for sufficient performance. For other training configurations, we use the same as those used in our experiments with the proposed architecture.

## C.2. Training SynAD

The E2E AD model consists of perception (tracking, mapping), prediction (motion forecasting, occupancy prediction), and planning modules. Existing models include those where each module is serialized [12] and those where they are configured in parallel [28]. To appropriately train the E2E AD model using two different types of inputs, multi-camera and map inputs, we suitably combine two design principles. The perception module detects and tracks each vehicle and pedestrian from the multi-camera input and recognizes the map composition through segmentation. In cases where map input is provided instead of multi-camera input, the perception module does not need to operate. Therefore, we design the perception module to operate independently by configuring it in parallel. In the prediction module, motion forecasting uses only BEV features as input, while occupancy prediction uses the BEV features and the output of motion forecasting, $\hat{q}_{\text{motion}}$, as inputs. The planning module uses only BEV features as input during training, and during inference time, it uses the output of occupancy prediction, $\hat{O}$, along with test-time optimization to reduce the collision rate.

**Module Configurations.** *The motion forecasting module* takes three embeddings($E_{\text{motion}}$, $E_{\text{agent}}$, $E_{\text{ego}}$) along with the BEV feature as inputs. The BEV feature has a channel dimension of 256 and spatial dimensions of $200 \times 200$. $E_{\text{motion}}$, $E_{\text{agent}}$, $E_{\text{ego}}$ each have shapes $M \times N \times 256$, where $M$ is the scenario-dependent number of agents and $N = 6$ represents the number of possible paths. These embeddings are initialized with learnable parameters of size 100, then sliced according to the number of agents in each scenario. The queries $\hat{q}_{\text{motion}}$ and $q_{\text{motion}}$ share the same shapes as their embeddings. The module outputs $\hat{\mathbf{x}} \in \mathbb{R}^{M \times N \times T \times 2}$, where $T = 12$. *In the occupancy prediction module*, $\hat{q}_{\text{motion}}$ passes through an MLP layer to form a tensor of shape $M \times 256$, and we also define $q^t_{\text{temp}} \in \mathbb{R}^{M \times 256}$. *In the planning module*, $q_{\text{plan}}$ and $\hat{q}_{\text{plan}}$ each have shape $1 \times 256$, and $B_a$ retains the same dimensions as $B$.

**Loss Configurations.** When $\lambda_{\text{motion}}$, $\lambda_{\text{occ}}$, and $\lambda_{\text{plan}}$ in Equation 23 are all set to 1.0, the detailed loss coefficients for each module are as follows: The loss coefficients $\lambda_{\text{JNLL}}$ and $\lambda_{\text{minFDE}}$ in the motion forecasting are set to 0.5 and 0.25, respectively. The loss coefficients $\lambda_{\text{dice}}$ and $\lambda_{\text{bce}}$ in the occupancy prediction are set to 1.0 and 5.0, respectively. In the planning, we used three values for $\delta$: 0, 0.5, and 1.0. Accordingly, the loss coefficients $\lambda_\delta$ are set to 2.5, 1.0, and 0.25, respectively.

**Hyperparameter Configurations.** We use the AdamW optimizer with a learning rate of $2 \times 10^{-4}$ and a weight decay of 0.01. We utilize a cosine annealing learning rate scheduler with a warm-up phase lasting for the initial 500 iterations at a ratio of one-third. We employ a cosine annealing learning rate scheduler, performing warm-up for the first 500 iterations at a ratio of one-third, and set the minimum learning rate to $2 \times 10^{-7}$. The momentum parameters are set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

# D. Reporting Rules for Other Works.

In Table 1, we evaluate our generated paths using the CTG++ [37] metrics, and the baseline performances are taken from the same reference. In Table 2, the reported results for UniAD [12], VAD, and ParaDrive [28] come from the most recent ParaDrive paper. For OccNet [26] and OccWorld [36], we use the performance as reported in OccWorld, selecting the best reported results. In Table 3, we obtain the performances of UniAD and VAD via their official codebases, and we rely on our own implementation for ParaDrive due to the lack of publicly available code. For the same reason, we adopt the VAD evaluation protocol for planning.