MDG: Masked Denoising Generation for Multi-Agent Behavior Modeling in Traffic Environments

Supplementary Material

A. Model Details

A.1. Scene Encoder

Agent Encoder. The agent state tensor contains per-timestep states for all agents, including their (x, y) coordinates, heading, velocity, and bounding box dimensions (length, width, height). All agent trajectories are transformed into their respective local coordinate frames, using the final observed state as the origin. An MLP-Mixer encoder is employed to capture both temporal dynamics and feature interactions within each agent's state sequence. Specifically, the agent state tensor is processed by stacked MLP-Mixer blocks that alternately mix information along the temporal and feature dimensions, producing a latent tensor of shape $[N, H, D]$, where N is the number of agents, H the historical timesteps, and D the hidden dimension. Temporal max-pooling is then applied to summarize motion patterns across time. Each agent type is embedded via a learnable embedding vector and added to its feature representation. The final agent encoding has the shape $[N, D]$.

Map Encoder. The map tensor consists of N_m polylines (e.g., road centerlines, lane boundaries, crosswalks), each with N_w waypoints described by (x, y) coordinates and heading. All map elements are transformed into local coordinates using the first waypoint of each polyline as the origin. Each polyline is processed by an MLP-Mixer that models both intra-polyline spatial dependencies and cross-feature correlations. The Mixer operates on a tensor of shape $[N_m, N_w, D]$, and its outputs are aggregated by max-pooling along the waypoint axis to produce a polyline-level feature tensor $[N_m, D]$. Polyline type and associated traffic-signal state are encoded via learnable embeddings and added to the feature representation. The final map encoding has the shape $[N_m, D]$.

Ego-Route Encoder. For planning tasks, we include route polylines for the ego agent, consisting of N_r polylines, each with N_w waypoints defined by (x, y) coordinates and heading. All waypoints are transformed into local coordinates using the first waypoint of each polyline as the origin. An MLP-Mixer encoder processes these route polylines in the same manner as the map encoder, capturing both waypoint-wise spatial relations and inter-feature dependencies. The resulting ego-route encoding has shape $[N_r, D]$.

Traffic Signal Encoder. Traffic signals are represented by their stop points. Since we adopt relative spatial encoding, only the current signal phase (e.g., red, yellow, green) is encoded. An MLP Embedding layer converts each signal state into a feature tensor of shape $[N_s, D]$.

Relative Relation Encoder. To model spatial relationships among all scene elements, we compute pairwise relative attributes between every pair (i, j). The relative distance and heading are computed as follows: (1) for agents, using the last observed position; (2) for map elements, using the first waypoint of each polyline; and (3) for traffic signals, using the stop point position. Self-relations are assigned a small constant. An MLP-based Fourier Embedding relation encoder [11] encodes these pairwise relations into a relation tensor of shape $[N + N_m + N_s, N + N_m + N_s, D]$, enabling subsequent attention modules to incorporate spatial context and topology-aware reasoning.

Query-centric Transformer Encoder. The Transformer encoder is employed to extract relationships among all scene elements. It comprises a stack of query-centric attention layers, which use the same attention mechanism in Eq. (S1). The relative relation encodings are incorporated into this encoder. All scene elements (agents, map polylines, and traffic signals) are concatenated into a tensor of shape $[N+N_m+N_s, D]$, which is input to the Transformer. After passing through several attention layers, the Transformer learns to capture the interactions and relationships between the scene elements. Invalid elements are masked out during the attention calculations. The final scene context encoding retains its original shape $[N + N_m + N_s, D]$. The relative position and heading differences between elements i and j are encoded as edge attributes, forming a relation encoding tensor $\mathbf{e}^{i\to j}$. The relative cross-attention (RCA) is defined as:

$$RCA(Q^i, K, V, \mathbf{e}) = \text{softmax}\left(\frac{\mathbf{q}^i}{\sqrt{D}}\left[\left\{\mathbf{k}^j + \mathbf{e}^{i\to j}\right\}_{j\in\Omega(j)}\right]^T\right) \times \left(\left\{\mathbf{v}^j + \mathbf{e}^{i\to j}\right\}_{j\in\Omega(j)}\right), \tag{S1}$$

where $\mathbf{q}^i, \mathbf{k}^j, \mathbf{v}^j$ represent the query, key, and value elements respectively, each containing relevant element-centric information, and $j \in \Omega(j)$ defines the set of indices corresponding to neighboring elements.

A.2. Decoder

Noised Future Encoder. We first compute each agent's control actions (acceleration and yaw rate) from their logged trajectories. These actions are normalized to form a clean action tensor of shape $[N, T_a, 2]$, where T_a denotes the reduced temporal dimension obtained via action chunking, which improves computational and memory efficiency. Gaussian noise is then applied to the clean actions, which

are subsequently transformed into noisy physical states (positions, headings, and velocities) through a differentiable dynamics function, resulting in a tensor of shape $[N, T_a, 5]$. The noised states are encoded using an MLP. Mask and timestep embeddings are then added to produce the noised future query tensor of shape $[N, T_a, D]$, serving as the input to the denoising Transformer.

Transformer Denoiser. The denoiser consists of stacked Transformer blocks, each containing complementary attention layers that operate over different relational dimensions:

- **Intra-Agent Temporal Attention.** A multi-head self-attention layer extracts temporal dependencies within each agent's future trajectory.
- **Inter-Agent Interaction Attention.** The outputs are passed through a cross-attention layer that models inter-agent interactions. Invalid or missing agents are masked. Relation encodings from the encoder are used as positional and relational priors (inter-agent relations) in this attention computation.
- **Agent-Scene Condition Cross-Attention.** A cross-attention layer fuses the agent features with scene context representations from the encoder (map, traffic lights, and agents). For planning tasks, the ego agent receives additional conditioning through cross-attention with the ego-route encoding, which provides route-level geometric and semantic guidance. This route information is exclusively accessible to the ego agent. Relation encodings for agents and scene elements (map polylines, traffic agents, traffic lights, and optional ego-route polylines) are incorporated into this attention process.

Residual connections are applied across all Transformer layers to stabilize optimization and preserve gradient flow. The final denoised latent tensor is decoded with an MLP that outputs denoised action sequences for each agent.

Trajectory Decoder. An auxiliary MLP decoder operates directly on the agent-specific scene context encodings to predict future trajectories of shape $[N, M, T, 3]$, where M is the number of trajectory modalities. This auxiliary prediction branch stabilizes training and encourages the scene encoder to learn trajectory-relevant representations.

Differentiable Dynamic Function. We adopt the differentiable dynamics function from [2] to convert predicted agent actions (acceleration and yaw rate) into corresponding physical states (positions and headings), conditioned on each agent's current state. As this function is differentiable, it is integrated into the model as a trainable layer, enabling end-to-end gradient propagation through both the kinematic transformation and the denoising process.

A.3. Model Parameters

The primary parameters used in our MDG model are summarized in Tab. S1 and Tab. S2. Separate configurations are adopted for the Waymo and nuPlan experiments to account for differences in scene complexity and task requirements. Unless otherwise specified, parameters in the nuPlan setup are identical to those in the Waymo configuration.

B. Training Details

The training objective combines two loss components and invalid agents and timesteps are excluded:

$$\mathcal{L} = \mathcal{L}_d + \lambda \mathcal{L}_p, \tag{S2}$$

where $\mathcal{L}_d$ represents the denoising loss, and $\mathcal{L}_p$ is an auxiliary prediction loss, and $\lambda = 5$ is the balance weight. The prediction loss $\mathcal{L}_p$ is defined as:

$$\mathcal{L}_p = \frac{1}{N} \sum_{i=1}^{N} \mathcal{SL}_1(\hat{\mathbf{s}}_i^* - \mathbf{s}_i^{gt}),\ i^* = \arg\min_m \|\hat{\mathbf{s}}_i^m - \mathbf{s}_i^{gt}\|_2, \tag{S3}$$

where $\mathcal{SL}_1$ denotes the smooth L_1 loss applied to the best-predicted trajectory $\hat{\mathbf{s}}_i^*$, defined as the trajectory with the minimal L_2 from the ground truth $\mathbf{s}_i^{gt}$.

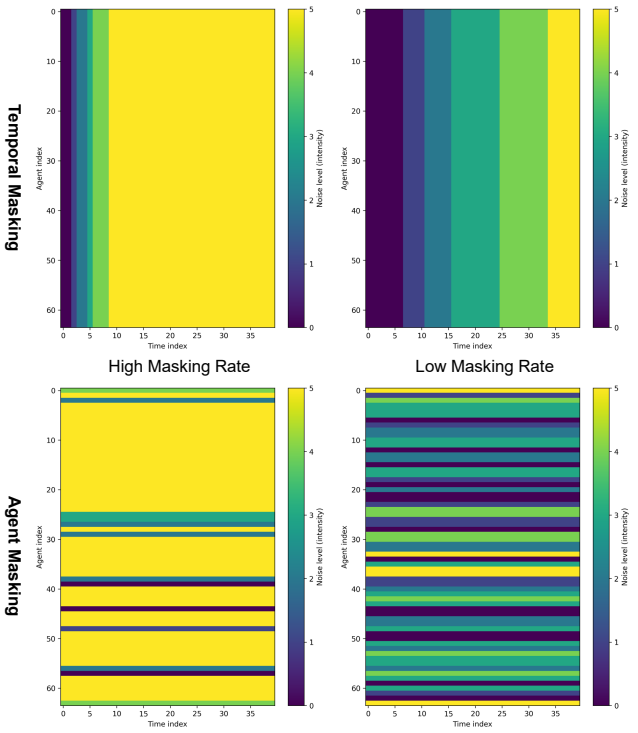

Figure S1. Illustration of the proposed training noise masking strategy. Top: Temporal masking; Bottom: Agent masking. Each column shows an example under either a high or low masking rate.

We propose a novel masking strategy that distributes the masking rate across training batches and applies masking randomly along either the temporal or agent axis. This approach facilitates adaptive denoising generation during inference and guidance. The training procedure is detailed

Table S1. Model Parameters for Waymo Experiments

Parameter	Value	Description
N	128	Number of agents per scene
N_m	320	Number of map polylines
N_w	16	Number of waypoints per polyline
N_s	16	Number of traffic lights
H	11	History steps
T	80	Future steps
T_a	40	Reduced future timesteps
D	256	Hidden dimension
M	6	Number of trajectory modalities in predictor
Modality encoder layers	2	Number of MLP-Mixer layers in the encoder
Encoder layers	6	Number of query-centric Transformer layers
Decoder blocks	2	Number of Transformer denoising blocks
Attention heads	8	Number of attention heads per layer
Dropout rate	0.1	Dropout probability
Feed-forward dimension	1024	Dimension of the feed-forward network
Action chunk	2	Action sequence granularity
Action mean	$[0.0, 0.0]$	Mean of action distribution [acceleration, yaw rate]
Action std	$[1.0, 0.5]$	Standard deviation of action distribution [acceleration, yaw rate]

Table S2. Model Parameters for nuPlan Experiments

Parameter	Value	Description
N	32	Number of agents per scene
N_m	256	Number of map polylines
N_m	32	Number of ego-route polylines
N_w	20	Number of waypoints per polyline
H	20	History steps
T	80	Future steps

in Algorithm 1. The strategy involves randomly selecting masking patterns: masking over time or masking over agents. For each training sample, a masking rate δ is assigned. When masking over time, a δ fraction of the later timesteps for each agent are fully noised, while other timesteps are randomly assigned noise levels that progressively increase. When masking over agents, a δ fraction of the agents are fully noised with the highest noise levels, and the remaining agents are randomly assigned lower noise levels, while the noise levels for an individual agent remain consistent across timesteps. Importantly, the masking rate δ is evenly distributed across all samples in a batch to ensure robust learning. Fig. S1 illustrates both masking types under high and low masking rates.

For Waymo experiments, to balance computational efficiency and model performance, we limit training to the 64 agents nearest to the labeled self-driving car (including itself). This reduces GPU memory usage while accommodating most scenarios, as they typically involve fewer than 64 valid agents. During testing, the model can scale up to 128 agents, leveraging the flexible attention mechanisms of the Transformer architecture.

For the nuPlan experiments, following established practice [1, 9], we apply random perturbations to the current state as data augmentation. A short interpolation segment is inserted to ensure a physically plausible transition, allowing the model to remain robust to perturbations and converge back to the ground-truth trajectory.

Training is performed on eight NVIDIA L40S GPUs using the AdamW optimizer with a weight decay of 0.01 and BFloat16 precision. The set of training hyperparameters is provided in Tab. S3.

C. Inference Details

Multi-step Denoising. Our proposed MDG framework offers flexibility during inference by enabling multi-step generation along either the temporal axis or the agent axis. Below, we elaborate on the inference process of our masked denoising model. The inference begins by initializing the process with full Gaussian noise, corresponding to a mask $\bar{\mathbf{m}}_L$ that fully covers the data with noise. This noisy in-

Table S3. Hyperparameters for Model Training

Hyperparameter	Value	Description
Noise levels	5	Number of distinct noise levels
Learning rate	0.0002	Initial learning rate
LR decay step	2000	Step interval for learning rate decay
LR warmup step	1000	Warmup steps at the beginning
LR decay factor	0.98	Multiplicative decay factor
Batch size	4	Number of samples per GPU
Training epochs	20	–
Gradient clip	1.0	Maximum gradient norm for clipping

Algorithm 1 Training Procedure for MDG

Input: Data X, Noise schedule α, Noise level K, Denoising model $\mathcal{D}_\theta$,
Output: Trained model $\mathcal{D}_{\theta^*}$
Initialize model parameters θ
for Each training batch X_b **do**
  for Each sample $x \in X_b$ **do**
    Randomly select masking type: time-axis or agent-axis
    Assign masking rate $\delta \in [0, 1]$
    if masking over time **then**
      Apply full noise to δ fraction of later timesteps
      Assign random progressively increasing noise levels to remaining timesteps
    else
      Apply full noise to δ fraction of agents
      Randomly assign noise levels across timesteps for each agent
    end if
    Generate noise mask $\mathbf{m}$ and add noise to x according to schedule $\alpha(\mathbf{m})$
  end for
  Perform a forward pass of the model on X_b
  Compute loss $\mathcal{L}$
  Backpropagate the loss and update θ using an optimizer
end for

Algorithm 2 Inference Procedure for MDG

Input: Denoising step L, Noise mask schedule $\{\bar{\mathbf{m}}_\ell\}_{\ell=L}^{0}$, Noise schedule α, Denoising model $\mathcal{D}_\theta$
Output: Denoised sample $\hat{\mathbf{x}} = \mathbf{z}_0$
Initialize $\mathbf{z}_L \sim \mathcal{N}(0, \mathbf{I})$ {Start with full Gaussian noise}
for $\ell = L$ to 1 **do**
  $\hat{\mathbf{x}}_\ell \leftarrow \mathcal{D}_\theta(\mathbf{z}_\ell, \bar{\mathbf{m}}_\ell)$ {Denoise the current sample}
  $\mathbf{z}_{\ell-1} \leftarrow \sqrt{\alpha(\bar{\mathbf{m}}_{\ell-1})}\hat{\mathbf{x}}_\ell + \sqrt{1-\alpha(\bar{\mathbf{m}}_{\ell-1})}\epsilon,\ \epsilon \sim \mathcal{N}(0, \mathbf{I})$ {Re-noise for next step}
end for
Return $\mathbf{z}_0$ {Final denoised output}

put, along with the corresponding mask, is fed into the denoising model to generate an intermediate clean sample. In the subsequent iteration, the generated clean sample is renoised according to the noise mask at the next step $\bar{\mathbf{m}}_{L-1}$. This noised sample is then passed through the model for denoising. This iterative process continues until the final step, where the final denoised sample is obtained. The complete inference procedure is illustrated in Algorithm 2.

The design of denoising schedules at inference time is important. We propose three strategies for this purpose. 1) *Single-step Denoising:* The model produces a clean output in a single pass starting from a fully masked noisy input. 2) *Temporal Axis Denoising:* Denoising proceeds progressively along the temporal dimension, analogous to autoregressive next-step generation. The noise level at each step is adjusted based on the remaining timesteps and the predefined noise scale, enabling flexible and fine-grained temporal refinement. 3) *Agent Axis Denoising:* Similar to temporal-axis denoising, this strategy iterates along the agent dimension instead. It offers flexibility in the ordering and progression of agents, which can be advantageous in multi-agent scenarios with heterogeneous importance or interaction structures. Fig. S2 illustrates an example of the temporal-axis inference schedule. These flexible strategies allow the MDG framework to tailor its denoising process to diverse tasks and operational constraints.

Guidance Imposition. The training mechanism of the MDG model facilitates a straightforward yet effective guidance imposition method to change the behavior of selected target agents while preserving the reactivity of other agents and maintaining overall scene consistency. This method involves replacing the denoised trajectories of the target agents with modified ones and adding small controlled noise. The adjusted trajectories are then fed into the next denoising iteration. To implement this, we define a guidance noise mask $\bar{\mathbf{g}}$, assigning a fixed low noise level $\alpha = 0.8$ to all timesteps of the target agents while leaving other agents

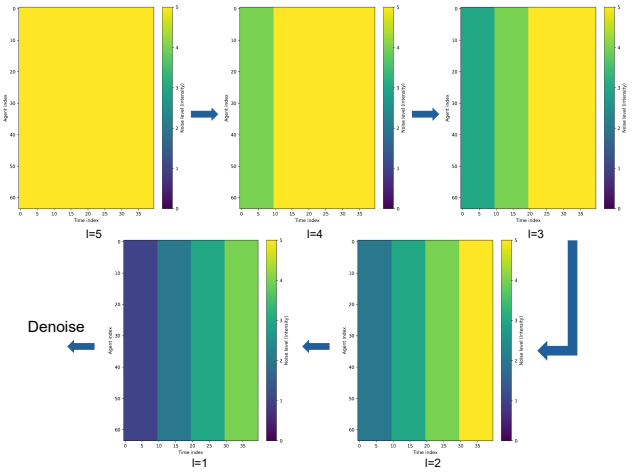

Figure S2. Example of a five-step temporal-axis denoising schedule used during inference.

unaffected. This mask remains consistent throughout the denoising generation process, and the noise mask schedule controls the denoising of other agents. This design allows us to explicitly optimize the trajectories of the target agents or replace them with prior prediction results, without compromising the overall scene dynamics. The guidance procedure is illustrated in Algorithm 3. Notably, unlike diffusion-based models [3, 10], our method avoids differentiating through the denoiser, which shows significantly faster inference speeds, enhancing its practical applicability.

Algorithm 3 Guidance Imposition Procedure for MDG

Input: Noise mask schedule $\{\bar{\mathbf{m}}_\ell\}_{\ell=L}^0$, Guidance noise mask $\bar{\mathbf{g}}$, Denoising schedule α, Denoising model $\mathcal{D}_\theta$, Objective function $\mathcal{J}$
Output: Denoised sample $\hat{\mathbf{x}} = \mathbf{z}_0$
Initialize $\mathbf{z}_L \sim \mathcal{N}(0, \mathbf{I})$ {Start with full Gaussian noise}
for $\ell = L$ to 1 **do**
  $\hat{\mathbf{x}}_\ell \leftarrow \mathcal{D}_\theta(\mathbf{z}_\ell, \bar{\mathbf{m}}_\ell)$ {Denoise the current sample}
  $\hat{\mathbf{x}}_\ell^M \leftarrow \mathcal{J}(\hat{\mathbf{x}}_\ell)$ {Modify the target agents' trajectories on the clean sample}
  $\mathbf{z}_{\ell-1} \leftarrow \sqrt{\alpha(\max(\bar{\mathbf{m}}_{\ell-1}, \bar{\mathbf{g}}))}\hat{\mathbf{x}}_\ell^M + \sqrt{1-\alpha(\max(\bar{\mathbf{m}}_{\ell-1}, \bar{\mathbf{g}}))}\epsilon$, $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ {Apply guidance mask and re-noise for the next step}
end for
Return $\mathbf{z}_0$ {Final denoised output}

D. Experiment Details

D.1. Evaluation Metrics

The evaluation metrics employed in the Waymo Sim Agents benchmark are detailed in [5]. These metrics are designed to quantify the distributional divergence between ground-truth agent trajectories and simulated agent trajectories. The evaluation contains three key aspects: kinematic features, interactive features (e.g., time-to-collision (TTC), collision rate, distance to the nearest object), and map-based features (e.g., off-road and distance to road edge). A meta-composite realism score aggregates these components and serves as the primary evaluation metric.

For the nuPlan benchmark, we employ the closed-loop evaluation metrics defined in [4]. These include No At-Fault Collisions, Drivable Area Compliance, Making Progress, Driving Direction Compliance, TTC Within Bound, Progress Along Route Ratio, Speed Limit Compliance, and Comfort. The individual scores are weighted to produce the final Closed-Loop Score (CLS).

In addition to these benchmark-specific metrics, we report several supplementary measures to compare our model against state-of-the-art methods in open-loop settings. These metrics include:

Collision Rate (CR). This metric measures the average collision rate per scene (the number of colliding agents divided by the total number of agents), averaged by the total number of testing scenarios:

$$\text{CR} = \frac{1}{N_\text{T}} \sum_{N_\text{T}} \frac{1}{N_\text{S}} \frac{1}{N_\text{A}} \sum_{j=1}^{N_\text{S}} \sum_{i=1}^{N_\text{A}} \mathbf{1}_c(s_i^j), \tag{S4}$$

where N_T is the number of test scenarios, N_S is the number of samples, N_A is the total number of modeled agents in a scenario, and s_i^j is the trajectory of agent i of sample j. $\mathbf{1}_c(s_i^j)$ is an indicator function that equals 1 if the simulated trajectory results in a collision, and 0 otherwise.

Off-road Rate (OR). This metric quantifies the proportion of agents deviating off-road. It is calculated as the average off-road rate per scene (samples and agents), averaged by the total number of test scenarios:

$$\text{OR} = \frac{1}{N_\text{T}} \sum_{N_\text{T}} \frac{1}{N_\text{S}} \frac{1}{N_\text{A}} \sum_{j=1}^{N_\text{S}} \sum_{i=1}^{N_\text{A}} \mathbf{1}_o(s_i^j), \tag{S5}$$

where $\mathbf{1}_o(s_i^j)$ is an indicator function that equals 1 if the simulated trajectory veers off-road (e.g., outside the road boundary), and 0 otherwise. Note that we exclude those agents that are already off-road or labeled as pedestrians.

Scene Average Displacement Error (SADE). SADE computes the average Euclidean distance (L2 norm) between logged ground-truth trajectories and simulated trajectories

Table S4. Influence of the auxiliary trajectory predictor on closed-loop performance

Task	**Predictor**	**Main Metric (↑)**	**minADE (m) (↓)**
Waymo Sim Agents	✓	**0.7842** (Realism Meta)	**1.3123**
	–	0.7682	1.4117
nuPlan Val14-NR	✓	**89.75** (Closed-loop Score)	–
	–	86.90	–

Table S5. Influence of action-to-state conversion on open-loop multi-agent trajectory prediction

Method	CR (%) ↓	OR (%) ↓	SADE (m) ↓	minSADE (m) ↓
W/ conversion	**4.875**	**3.274**	**3.122**	**1.840**
W/o conversion	8.875	6.066	4.353	2.853

across all agents and scenarios:

$$\text{SADE} = \frac{1}{N_\text{T}} \sum_{N_\text{T}} \frac{1}{N_\text{S}} \frac{1}{N_\text{A}} \sum_{j=1}^{N_\text{S}} \sum_{i=1}^{N_\text{A}} \|s_i^j - s_i^{gt}\|_2, \quad \text{(S6)}$$

where s_i^{gt} represents the ground-truth trajectory of an agent.

Minimum SADE (minSADE). This metric captures the minimum SADE (averaged over objects and minimum over samples), providing a measure of the best-case alignment with ground truth.

$$\text{minSADE} = \frac{1}{N_\text{T}} \sum_{N_\text{T}} \min_j \frac{1}{N_\text{A}} \sum_{i=1}^{N_\text{A}} \|s_i^j - s_i^{gt}\|_2. \quad \text{(S7)}$$

Goal-reach Rate (GR). This metric measures the average number of agents that successfully reach their assigned goals across all scenarios. An agent is considered to have reached its goal if the distance between its position and the goal position is less than 1 meter. The final value is obtained by averaging over the total number of scenarios.

D.2. Baseline Methods for Open-loop Tasks

SMART [7] is a state-of-the-art autoregressive next-token generation model for traffic agent simulation, achieving top performance on the Waymo Sim Agents benchmark. In our experiments, we use the tiny variant of SMART with 7M parameters. We adopt the open-source implementation provided in CAT-K [8] and follow the default training and inference settings. To ensure a fair comparison, we do not perform closed-loop fine-tuning and only use the first-stage behavior cloning training.

VBD [2] is a diffusion-based generative model for traffic agent simulation and ranked second in the 2024 Waymo Sim Agents Challenge. VBD performs joint trajectory diffusion, where all agents at a given diffusion step share the same noise level. We reproduce the model using the authors' open-source implementation and evaluate it with five diffusion steps, generating the final predictions under this configuration.

MTR [6] is a high-performance motion prediction model achieving state-of-the-art results on the Waymo Motion Prediction benchmark. We adapt MTR for multi-agent simulation by using the same encoder as in our model and extending the decoder to produce predictions for all agents simultaneously. Predefined trajectory anchors serve as initial queries for each agent, allowing marginal predictions for all agents in a single forward pass without iterative querying.

E. Additional Results

E.1. Ablation Studies

Influence of the Auxiliary Predictor. To examine the effect of the auxiliary trajectory predictor in the model, we conduct an ablation study by removing the predictor head from the model. As summarized in Tab. S4, excluding the predictor leads to a consistent decline in performance across both benchmarks. On the Waymo Sim Agents evaluation, the realism meta-metric drops from 0.7842 to 0.7682, while minADE increases from 1.3123 m to 1.4117 m, reflecting degraded simulation realism and motion accuracy. Similarly, on the nuPlan Val14 benchmark, CLS decreases from 89.75 to 86.90, indicating reduced planning performance. These results demonstrate that the auxiliary predictor enhances representation learning within the scene encoder by providing trajectory-level supervision.

Influence of Action-to-State Conversion. The Transformer-based denoiser operates on a spatiotemporal representation of agent behaviors. To better capture motion dynamics, a differentiable kinematic function converts agent actions (e.g., acceleration and yaw rate) into corresponding physical states (x, y, θ, v), which are then encoded and denoised. We evaluate the impact of this action-to-state conversion on multi-agent open-loop trajectory prediction. All models are trained with five noise levels and evaluated using five-step temporal denoising

Table S6. Influence of model scale on open-loop multi-agent trajectory prediction

Scale	# Model Params	CR (%) ↓	OR (%) ↓	SADE (m) ↓	minSADE (m) ↓
D=128	2.5M	5.477	4.585	3.714	2.358
D=256	10.0M	**4.875**	**3.274**	**3.122**	**1.840**
D=512	39.4M	4.905	3.425	3.289	1.921

Table S7. Ablation on attention mechanisms in the MDG denoiser on open-loop multi-agent trajectory prediction

Attention Modules			**Metrics**		
Intra-Agent	Inter-Agent	Agent-Scene	CR (%) ↓	SADE (m) ↓	minSADE (m) ↓
✓	✓	✓	**4.875**	**3.122**	**1.840**
✓	–	✓	6.821	3.580	2.060
–	✓	✓	5.612	3.460	1.990
–	–	✓	9.964	4.070	2.310

to ensure consistent settings. As shown in Tab. S5, integrating the conversion module substantially improves all evaluation metrics. Specifically, incorporating physical state transformation reduces collision and off-road rates by more than 40% and decreases both SADE and minSADE by a significant margin. This improvement highlights the importance of embedding kinematic consistency within the denoiser: the conversion enables the model to reason over sequential dynamics rather than isolated action tokens, resulting in more physically coherent and accurate trajectory generation.

Influence of Model Scale. We examine the impact of model scale on open-loop prediction performance by varying the hidden dimension of the Transformer while keeping the number of layers fixed. All models are trained for the same number of epochs, with batch sizes adjusted to fit GPU memory constraints. As shown in Tab. S6, enlarging the hidden dimension from 128 to 256 consistently improves performance across all metrics. However, further scaling to 512 yields no additional gains, indicating diminishing returns. This saturation may stem partly from smaller batch sizes used at larger scales, but the primary factor is likely the limited training data, which restricts the model's ability to fully leverage its increased capacity.

Influence of Attention in Denoiser. We analyze the contribution of each attention module in the Transformer denoiser. As reported in Tab. S7, removing any of the three components consistently degrades prediction performance across all metrics, while the full model achieves the best overall results. These findings highlight the complementary roles of intra-agent, inter-agent, and agent-scene attention in the denoiser.

E.2. Visualization

Denoising Process. We provide visualizations of the denoising process to illustrate its behavior. Fig. S3 depicts the temporal denoising process over five steps. Initially, the denoising results exhibit similarities across predictions. However, as the process progresses, subsequent denoising steps yield increasingly distinct outcomes for future predictions. Temporal denoising primarily focuses on the agents' dynamic behaviors, and increasing the number of denoising steps enables greater variation in intermediate actions. This, in turn, enhances the behavioral diversity across the entire time horizon. In Fig. S4, we demonstrate the denoising process along the agents over five steps. Unlike temporal denoising, agent-axis denoising emphasizes iterative trajectory-level refinement. Early denoising results are suboptimal for agents with high noise levels. However, as the denoising progresses, the trajectories become increasingly refined. This iterative process leads to a gradual determination of agents' behaviors, ultimately producing diverse and multi-modal joint agent trajectories.

Closed-loop Reuse. An example of the closed-loop reuse denoising method in the Waymo dataset is illustrated in Fig. S5. In the case of a simple one-step denoising method, the ego agent's planning results exhibit significant variability across consecutive frames, even with small intervals, indicating poor temporal consistency. In contrast, the closed-loop one-step reuse method demonstrates improved temporal consistency, yielding smoother planning trajectories and better overall planning performance.

Comparison on nuPlan Data. We compare MDG and the Diffusion Planner [9] in a representative nuPlan scenario, as shown in Fig. S6. The Diffusion Planner generates trajectories that lack temporal coherence across planning steps, causing the ego vehicle to drift from a safe course and ultimately collide with surrounding traffic. In contrast, MDG preserves temporal consistency through its closed-loop denoising mechanism, enabling stable motion planning and safe scenario completion.

Table S8. Detailed performance metrics of MDG across different benchmark splits

Benchmark	Score	Collision	TTC	Drivable area	Driving direction	Comfort	Ego progress	Speed limit
Val14 Non-Reactive	90.45	96.10	91.80	98.77	99.69	94.87	94.30	96.95
Val14 Reactive	83.89	93.54	88.33	98.33	99.58	90.01	86.58	97.82
Test14 Non-Reactive	90.16	95.93	91.46	98.78	99.69	94.51	94.08	96.85
Test14 Reactive	83.21	93.23	87.70	98.15	99.59	89.34	86.08	97.85

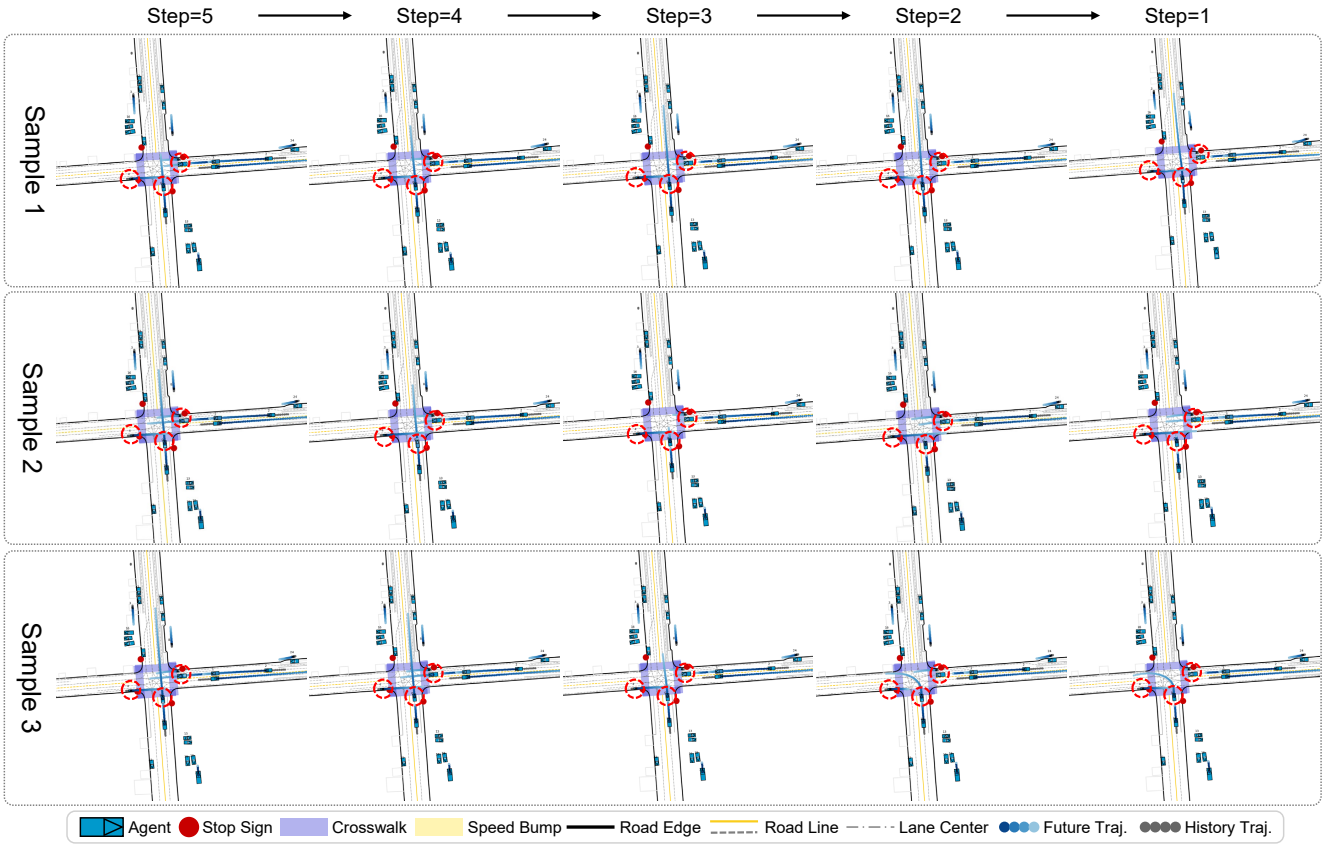

Figure S3. Visualization of the denoising process over time with five steps. Clean, denoised samples predicted by the MDG model at each step are shown. The process begins with similar predictions and gradually evolves to generate diverse future outcomes, emphasizing temporal behavior refinement and increased sample diversity across the time horizon.

E.3. Detailed Benchmark Results

Tab. S8 presents the detailed metrics of the MDG (1-step closed-reuse) method on the nuPlan benchmarks. Performance on reactive settings is lower primarily due to the simplistic IDM used for reactive agents, which produces unrealistic behaviors and artificial gaps.

References

[1] Jie Cheng, Yingbing Chen, and Qifeng Chen. Pluto: Pushing the limit of imitation learning-based planning for autonomous driving. *arXiv preprint arXiv:2404.14327*, 2024. 3

[2] Zhiyu Huang, Zixu Zhang, Ameya Vaidya, Yuxiao Chen, Jaime Fernández Fisac, and Chen Lv. Versatile behavior diffusion for generalized traffic agent simulation. *IEEE Transactions on Intelligent Transportation Systems*, 2026. 2, 6

[3] Chiyu Jiang, Andre Cornman, Cheolho Park, Benjamin Sapp, Yin Zhou, Dragomir Anguelov, et al. Motiondiffuser: Controllable multi-agent motion prediction using diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9644–9653, 2023. 5

[4] Napat Karnchanachari, Dimitris Geromichalos, Kok Seang Tan, Nanxiang Li, Christopher Eriksen, Shakiba Yaghoubi, Noushin Mehdipour, Gianmarco Bernasconi, Whye Kit Fong, Yiluan Guo, et al. Towards learning-based planning: The nuplan benchmark for real-world autonomous driving. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 629–636. IEEE, 2024. 5

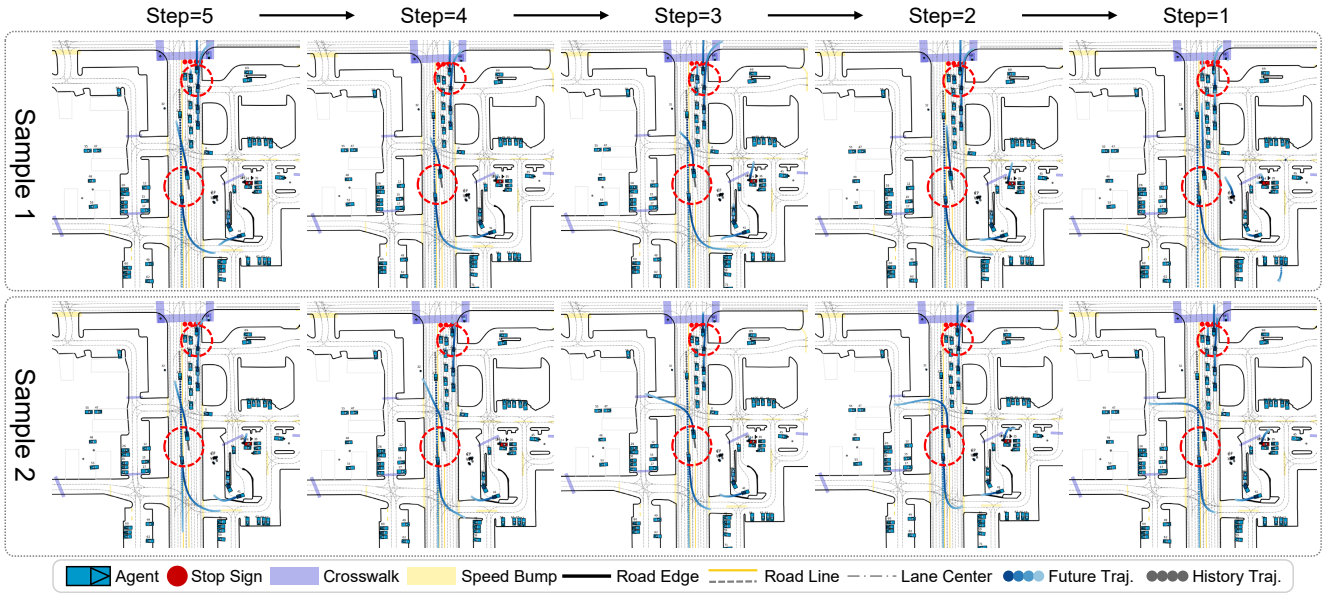

Figure S4. Visualization of the denoising process across agents over five steps. Clean, denoised samples predicted by the MDG model at each step are shown. Initial predictions for agents with high noise levels are suboptimal, but iterative refinement improves trajectory accuracy and enhances the diversity of behaviors.

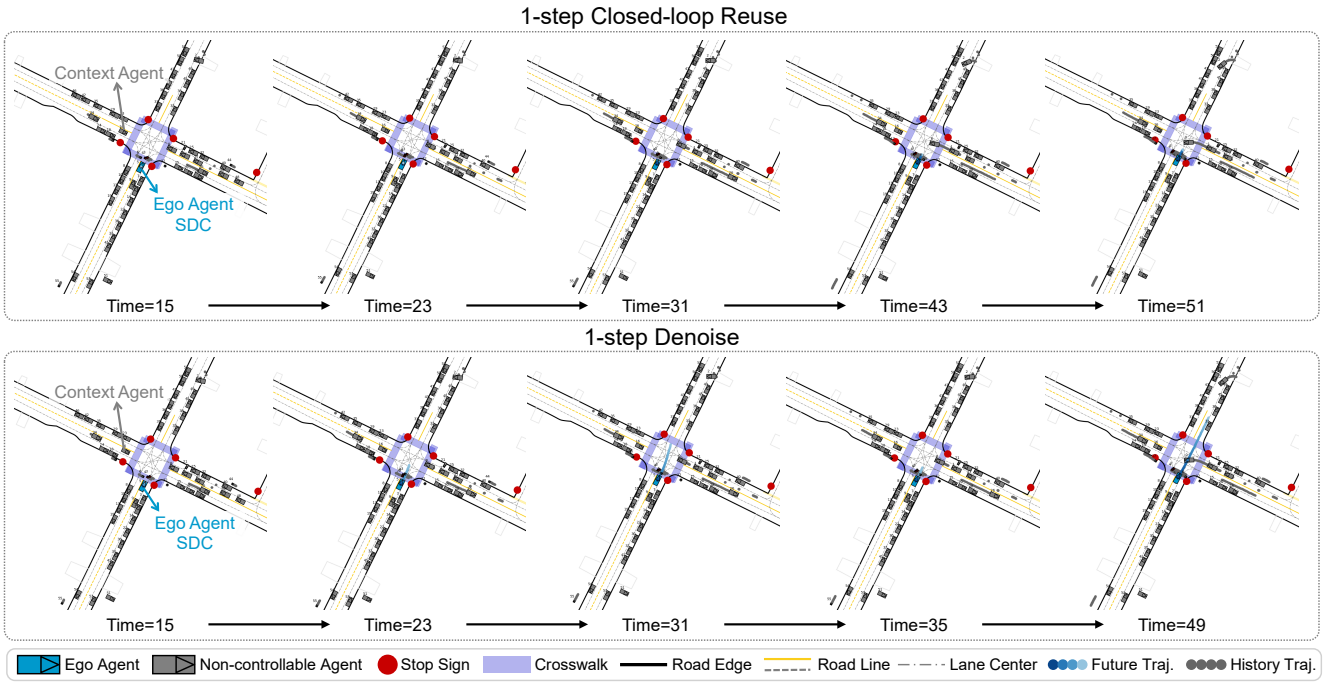

Figure S5. Comparison of a simple one-step denoising method and the closed-loop one-step reuse denoising method. The reuse method demonstrates improved temporal consistency and smoother planning trajectories for the ego agent.

[5] Nico Montali, John Lambert, Paul Mougin, Alex Kuefler, Nicholas Rhinehart, Michelle Li, Cole Gulino, Tristan Emrich, Zoey Yang, Shimon Whiteson, et al. The waymo open sim agents challenge. *Advances in Neural Information Processing Systems*, 36, 2024. 5

[6] Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. Motion transformer with global intention localization and local movement refinement. *Advances in Neural Information Processing Systems*, 35:6531–6543, 2022. 6

[7] Wei Wu, Xiaoxin Feng, Ziyan Gao, and Yuheng Kan. Smart:

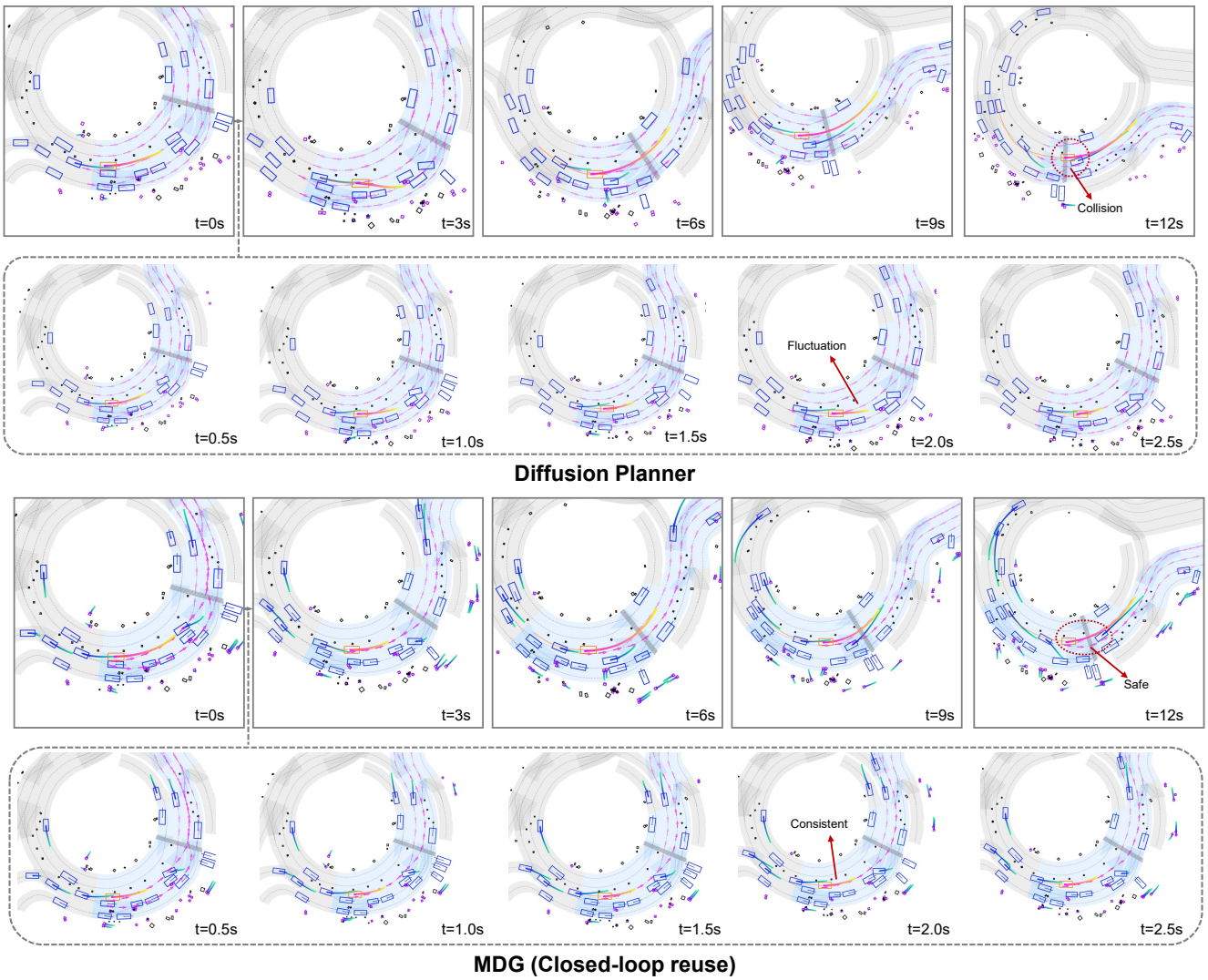

Figure S6. Comparison between MDG and Diffusion Planner in a nuPlan scenario. **Top:** The Diffusion Planner produces temporally inconsistent trajectories, leading to a collision. **Bottom:** MDG maintains consistent trajectories through a closed-loop denoising strategy and successfully completes the scenario without collisions.

Scalable multi-agent real-time motion generation via next-token prediction. *Advances in Neural Information Processing Systems*, 37:114048–114071, 2024. 6

[8] Zhejun Zhang, Peter Karkus, Maximilian Igl, Wenhao Ding, Yuxiao Chen, Boris Ivanovic, and Marco Pavone. Closed-loop supervised fine-tuning of tokenized traffic models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 5422–5432, 2025. 6

[9] Yinan Zheng, Ruiming Liang, Kexin ZHENG, Jinliang Zheng, Liyuan Mao, Jianxiong Li, Weihao Gu, Rui Ai, Shengbo Eben Li, Xianyuan Zhan, and Jingjing Liu. Diffusion-based planning for autonomous driving with flexible guidance. In *The Thirteenth International Conference on Learning Representations*, 2025. 3, 7

[10] Ziyuan Zhong, Davis Rempe, Danfei Xu, Yuxiao Chen, Sushant Veer, Tong Che, Baishakhi Ray, and Marco Pavone. Guided conditional diffusion for controllable traffic simulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3560–3566. IEEE, 2023. 5

[11] Zikang Zhou, Jianping Wang, Yung-Hui Li, and Yu-Kai Huang. Query-centric trajectory prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17863–17873, 2023. 1