

ColaVLA: Leveraging Cognitive Latent Reasoning for Hierarchical Parallel Trajectory Planning in Autonomous Driving

Supplementary Material

Visual tokens	L2 (cm) ↓			
	1s	2s	3s	Avg
Full tokens	14.8	28.9	52.4	32.0
Pruned tokens (Ours)	14.0	27.1	50.2	30.4

Table 1. Ablation on planner context token selection. Using the pruned critical tokens as context for our planner yields better accuracy while reducing sequence length and computational cost.

1. Further Ablation Study

Ablation on context tokens for planning. We further study whether the planner should attend to all visual tokens or only to the pruned subset selected by the ego-adaptive router. Concretely, we compare two variants: one that uses the full set of original vision tokens when predicting the final trajectory, and one that relies solely on the pruned critical tokens Q^* while keeping all other components unchanged. As summarized in Tab. 1, using pruned tokens leads to lower L2 errors compared to using all tokens. This indicates that the ego-adaptive router can accurately identify which visual cues are most relevant to the current driving scenario and future decisions, effectively filtering out redundant background information. At the same time, pruning substantially reduces the token length fed into the planner, thereby lowering computation and improving inference efficiency without sacrificing final prediction accuracy.

2. Further Implementation Details

Architecture. Our implementation is built on the LLaVA v1.5 framework [6], which uses LLaMA-7B as the language backbone and introduces LoRA adapters [4] for efficient adaptation. The image encoder follows EVA-02-L [3] instantiated as an EVAViT backbone with 24 transformer layers, hidden dimension 1024 and window size 16, initialized from publicly available weights. On top of the image backbone, we adopt the SQ-Former architecture [2] for multi-view visual reasoning and use perception task heads. We use 900 object queries and 300 lane queries by default, consistent with our main experiments. The overall system is implemented in a single camera-only configuration with 6 multi-view images per frame. And point-cloud-style BEV parameters (e.g., point cloud range and voxel size) are used only for defining the spatial grid for 3D perception heads.

Multi-stage training strategy. We employ a three-stage training pipeline to fully exploit the strong VLM priors while stabilizing downstream hierarchical planning:

- *Stage 1: VLM adaptation on OmniDrive QA.* We first adapt the vision-language backbone on OmniDrive-nuScenes QA pairs [9]. Similar to OmniDrive, we use scene-level QA covering description, perception, prediction and planning questions, and optimize only the LoRA layers on top of the LLaMA-7B language model together with the Q-Former style visual interface. This stage encourages the VLM to encode driving-relevant semantics and reasoning skills in complex urban scenes without changing the base model weights, and aligns the visual encoder with driving-oriented questions.
- *Stage 2: Meta-action bank pretraining.* In the second stage, we introduce the meta-action bank used by the Cognitive Latent Reasoner. We cluster nuScenes trajectories (using the provided k -means anchors in the config) to obtain a discrete set of meta-action prototypes, which are used to initialize the meta-action queries and the action bank. During this stage, we freeze the language backbone VLM (except LoRA), and primarily train the meta-action classification and trajectory prediction heads to associate latent meta-actions with typical driving patterns (e.g., straight, lane change, turn, braking). This provides a structured and stable prior for downstream hierarchical planning.
- *Stage 3: End-to-end fine-tuning on nuScenes.* Finally, we fine-tune ColaVLA on nuScenes planning supervision. The language backbone remains frozen while LoRA adapters, the visual encoder (EVAViT + SQ-Former), detection/map heads, cognitive latent reasoning components (router and meta-action queries), and the hierarchical parallel planner are all jointly optimized. We train with multi-scale trajectory losses and the top- K hypothesis strategy, which enables confidence-guided selection among multiple meta-action hypotheses.

Optimization and training schedule. Unless otherwise specified, all stages are trained on 16 NVIDIA H20 GPUs with a per-GPU batch size of 2 (effective batch size 32). We use the AdamW optimizer [7] with initial learning rate 1×10^{-4} , weight decay 1×10^{-4} , and $(\beta_1, \beta_2) = (0.9, 0.999)$. The learning rate is scheduled by Cosine Annealing [8] with linear warmup for the first 500 iterations and a minimum learning-rate ratio of 10^{-3} . We train for 10 epochs on nuScenes dataset in the final stage, using mixed-precision (FP16) with dynamic loss scaling and gradient clipping at a global norm of 10.

Data processing. We follow the standard nuScenes setting [1] with the official train/val split. Our data loader uses

CustomNuScenesDataset with temporal sequences, camera-only inputs, and the same object classes as the main experiments. Images are first loaded at 900×1600 resolution and then augmented with random resize–crop–rotation as in the provided augmentation configure. For training, we resize each view to 320×640 and normalize them using ImageNet statistics. We adopt the same pipeline for validation but disable random augmentation. Planning supervision is derived from nuScenes ego trajectories, while QA supervision and lane-object relations are loaded from the OmniDrive and preprocessed lane-object files.

Summary of key hyperparameters. For clarity, Tab. 2 summarizes the main architectural and training hyperparameters used in our implementation.

3. Additional Closed-Loop Visualizations

We further provide qualitative results from the NeuroNCAP simulator to complement the nuScenes open-loop visualizations in the main paper. As illustrated in Fig. 1, our model produces stable and collision-averse behaviors across diverse safety-critical scenarios, including static obstacles, frontal interactions, and side conflicts. Compared with previous text-based VLM planners, ColaVLA exhibits more responsive closed-loop reactions and fewer failure cases, while simultaneously maintaining higher inference efficiency. These examples corroborate our quantitative findings that ColaVLA achieves stronger NeuroNCAP scores and lower latency, delivering both safer and faster closed-loop planning.

4. Details about Meta-Action Bank

To support cognitive latent reasoning, we pre-compute a discrete meta-action label for each nuScenes sample and use it to index the action bank. Each meta-action is defined by combining a *path pattern* with a *speed pattern* derived jointly from the future ego trajectory and CAN bus signals. In our final setup, this yields eight meta-action classes: four straight-driving modes (stopping, constant-speed cruising, accelerating, and decelerating along a roughly straight path) and four turning or lane-change modes (large left turns or U-turns, large right turns or U-turns, small left turns or lane changes, and small right turns or lane changes). Our implementation broadly follows the trajectory categorization procedure used in Senna [5].

Geometry and CAN bus fusion. For each sample, we first obtain a future ego trajectory in the ego frame as a sequence of T waypoints at a fixed time interval. If the planning trajectory is missing or fully padded, we reconstruct it from nuScenes CAN bus pose messages by transforming global positions into the initial ego frame and resampling to T steps; if this is still not possible, we fall back to a zero trajectory. In parallel, we read CAN bus logs around the planning horizon (pose, yaw, longitudinal velocity and acceleration), and com-

pute fused heading and speed statistics such as cumulative yaw angle, maximum yaw rate, final speed, speed change, and mean acceleration. When available, CAN bus statistics are preferred; otherwise, we rely on geometric estimates from the trajectory itself.

Path classification. We first assign each sample to one of five path types: straight motion, large left turn (including left U-turns), large right turn (including right U-turns), small left turn or lane change, and small right turn or lane change. The assignment is determined using fused heading and lateral-motion statistics. Large turns are identified by large cumulative yaw with a clear left or right sign; small turns correspond to moderate yaw changes; lane-change-like motions are characterized by small net yaw but significant lateral displacement with consistent lateral direction and bounded yaw rate; straight motion is defined by small yaw and small lateral offset. Ambiguous residual cases are assigned to left or right small turns according to the sign of the lateral displacement. This step yields a path label for every sample.

Speed classification and heavy paths. Speed patterns are only refined for path types that appear frequently in the training set. We first count occurrences of each path type and mark *heavy* paths whose counts exceed a global threshold. In practice, only straight paths are heavy, while turning and lane-change paths are relatively sparse.

For heavy paths, we further classify the speed profile into four modes: stopping, approximately constant speed, accelerating, and decelerating. The decision uses fused speed statistics: very low final speed indicates stopping; clearly increasing speed or positive mean acceleration indicates acceleration; clearly decreasing speed or negative mean acceleration indicates deceleration; otherwise the sample is treated as constant-speed. For non-heavy path types, we do not split by speed and simply mark the speed pattern as “any”, in order to avoid severe class imbalance.

Final meta-action labels. The final meta-action class for each sample is defined as the combination of its path type and speed type. After applying the heavy-path and speed rules above, only eight combinations are active in training: four straight-driving modes (stop, constant-speed, accelerate, decelerate along a straight path) and four turning or lane-change modes (large left, large right, small left, small right). Each combination is assigned a stable integer id, which we precompute offline and store alongside each nuScenes sample. During training, the cognitive latent reasoner predicts a distribution over these eight meta-actions, and the top-scoring modes are used to retrieve corresponding action priors from the action bank, providing structured, trajectory-aligned guidance for the hierarchical parallel planner.

Category	Setting
Backbone VLM	LLaVA v1.5 (LLaMA-7B) with LoRA adapters
Image encoder	EVA-02-L (EVAViT, 24 layers, 1024-d, window size 16)
Visual reasoning	SQ-Former with temporal PETR-style transformers
Object queries	900 queries (3D detection head)
Lane queries	300 queries (map / lane head)
Input modality	6-camera views, camera-only, no LiDAR or radar
Image resolution	$900 \times 1600 \rightarrow 320 \times 640$ after augmentation
Point cloud range	$[-51.2, -51.2, -5.0, 51.2, 51.2, 3.0]$
Voxel size	$[0.2, 0.2, 8.0]$ (for BEV grid definition)
Planner stages	$S = 6$ hierarchical scales (as in <code>stage_index</code>)
Multi-scale loss	Weights $[0.5, 0.7, 1.0, 1.2, 1.5, 1.8]$
Top- K hypotheses	$K = 3$ meta-action modes (<code>topk_mode_predict=3</code>)
Batch size	2 samples / GPU ($\times 16$ GPUs)
Epochs (stage 3)	10 epochs on nuScenes
Optimizer	AdamW, lr 1×10^{-4} , wd 1×10^{-4} , (0.9, 0.999)
LR schedule	Cosine Annealing + 500-iter linear warmup, min lr ratio 10^{-3}
Precision	FP16 with dynamic loss scaling
Grad clipping	Global norm 10
Efficiency tricks	Gradient checkpointing, FlashAttention, mixed precision

Table 2. Main implementation and training hyperparameters for ColaVLA.

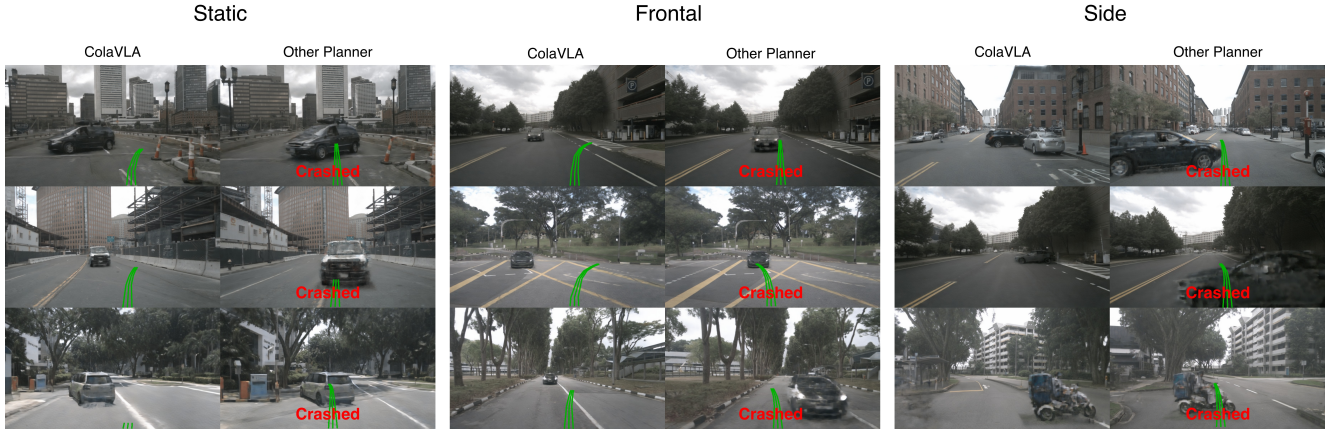


Figure 1. Qualitative closed-loop comparisons in the NeuroNCAP simulator across three representative scenario types. For each case, we visualize the predicted trajectories of ColaVLA and competing planners. ColaVLA consistently guides the ego vehicle away from potential collisions, producing safer and more stable motions.

References

- [1] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *CVPR*, 2020. 1
- [2] Xuesong Chen, Linjiang Huang, Tao Ma, Rongyao Fang, Shaoshuai Shi, and Hongsheng Li. Solve: Synergy of language-vision and end-to-end networks for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12068–12077, 2025. 1
- [3] Yuxin Fang, Quan Sun, Xinggang Wang, Tiejun Huang, Xinlong Wang, and Yue Cao. Eva-02: A visual representation for neon genesis. *Image and Vision Computing*, 149:105171, 2024. 1
- [4] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. 1
- [5] Bo Jiang, Shaoyu Chen, Bencheng Liao, Xingyu Zhang, Wei Yin, Qian Zhang, Chang Huang, Wenyu Liu, and Xinggang Wang. Senna: Bridging large vision-language models and end-to-end autonomous driving. *arXiv preprint arXiv:2410.22313*, 2024. 2
- [6] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *NeurIPS*, 36, 2024. 1
- [7] I Loshchilov. Decoupled weight decay regularization. In *ICLR*, 2019. 1

- 211 2019. [1](#)
- 212 [8] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient
- 213 descent with warm restarts. In *ICLR*, 2017. [1](#)
- 214 [9] Shihao Wang, Zhiding Yu, Xiaohui Jiang, Shiyi Lan, Min
- 215 Shi, Nadine Chang, Jan Kautz, Ying Li, and Jose M Alvarez.
- 216 Omnidrive: A holistic llm-agent framework for autonomous
- 217 driving with 3d perception, reasoning and planning. *arXiv*
- 218 *preprint arXiv:2405.01533*, 2024. [1](#)