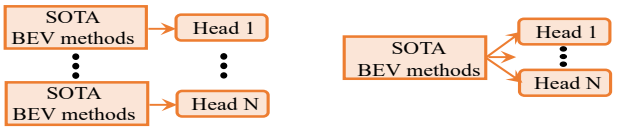
Holistic Autonomous Driving Understanding by Bird's-Eye-View Injected Multi-Modal Large Models

Supplementary Material



(a) Separate model **(b) Multi-head**

Figure 1. **(a)** Separate SOTA BEV methods. **(b)** Multi-head SOTA BEV methods.

1. comparison between ours and existing BEV SOTAs without LLMs

❶ Ours surpass BEV SOTAs in perception tasks with two benefits: **First**, our method can handle complex perception tasks involving reasoning or human interaction (`'What is the closest car near to <car>[675,475,994,755]'`, see more examples in Supplementary Material), while BEV SOTAs cannot. **Second**, our unified model efficiently addresses diverse tasks without task-specific training, unlike BEV SOTAs requiring separate training for each task (see Fig.1(b)-(c)). ❷ Even directly compared with specialized BEV SOTAs in Table 1, our model also shows competitive performance. BEV SOTAs will be integrated as tools in the LLM-based agent system for enhanced performance in our future work. ❸ Perception tasks are only a small part of our focus. Our main goal is to apply LLMs for more complex reasoning tasks (such as prediction, risk assessment, and planning), capabilities not offered by BEV SOTAs.

Table 1. **Comparison between our method and SOTA BEV methods without LLMs.** We use BEVFormer [3] as the BEV SOTA backbone. * indicates tasks requiring reasoning or interactions. ✗ means tasks can not be addressed by BEV SOTAs. All models are trained under the same setting.

Model	# Ins ↓	Clos ↑	*Perception**		*Prediction*		Risk ↑	Planning ↑
			↓	↑	↓	↑		
Ours (MiniGPT-4)	3.8	32.9	**13.3**	**25.4**	**4.2**	**46.5**	**22.0**	**35.6**
BEV SOTA[1](Fig. 1 (b))	3.2	33.7	✗	✗	✗	✗	✗	✗
BEV SOTA[2](Fig. 1 (c))	3.5	33.4	✗	✗	✗	✗	✗	✗

2. More details about NuInstruct

In this section, we give more information about our NuInstruct. In Section 2.1, we show the detailed information of the scene database. Then, we give the definition of Algorithms 1-17 for all task SQLs in Section 2.2. Finally, the computation for different metrics is presented in Section 2.3.

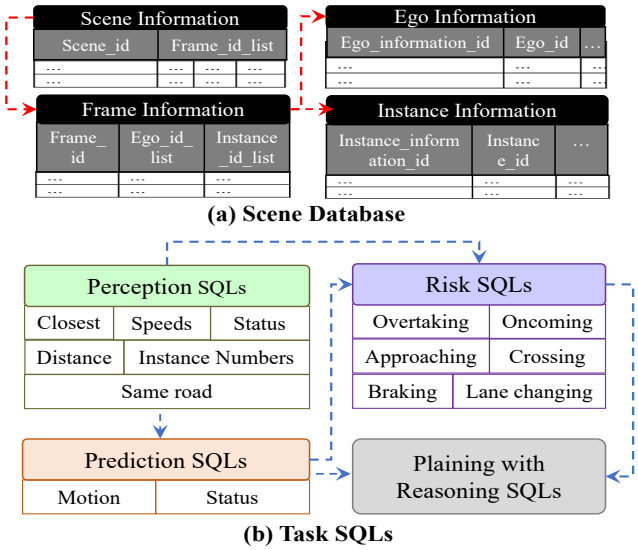


Figure 2. The illustration of **(a) Scene Database** and **(b) Task SQLs.** The red dashed line indicates the mapping relation of different tables. The red blue line indicates the derivation.

2.1. Scene Database

Based on the information of the ego car and important instances, we construct a scene database as shown in Fig. 2 (a). The scene information database includes four tables:

- `Scene Information`: consists of the scene dictionary list. The key for each dictionary is the scene ID, which is the unique identifier of the scene in NuScenes [1]. The value is a frame ID list, consisting of IDs of frames in the current scene. The details for each frame are referred to the `Frame Information` table.
- `Frame Information`: consists of the frame dictionary list. The key for each dictionary is the unique ID for a specific frame. The value contains the details for the frame, including the ego-car-information ID and instance-information ID list. The details for the information of the ego car and instances in the frame are referred to the `Ego Information` and `Instance Information` tables respectively.
- `Ego Information`: consists of the ego-car information dictionary list. The key for each dictionary is the unique ID for the information of the ego-car in one specific frame. The values include the information: *e.g.*,

Field	*Scene ID*	*Frame ID List*
Type	string	string list

(a) **Scene Information Table T_{scene}.**

Field	*Frame ID*	*Ego Information ID List*	*Instance Information ID List*
Type	string token	string list	string list

(b) **Frame Information Table T_{frame}.**

Field	*Information ID*	*Pose*	*Rotation*	*Velocity*	*Road Information*	*Camera Information*
Type	string	float list	float list	float	dictionary	dictionary

(c) **Ego Information Table T_{ego}.**

Field	*Information ID*	*Instance ID*	*Category*	*Attribute*	*Global-T*	*Global-R*	*Local-T*	*Local-R*	*Velocity*	*Road Information*	*Camera Pos*
Type	string	string	string	string	float list	float list	float list	float list	float	dictionary	dictionary

(d) **Instance Information Table T_{ins}.** Global-T = Global Translation, Global-R = Global Rotation, Local-T = Local Translation, Local-R = Local Rotation

Table 2. **Detaild Field and Type for different tables in** Fig. 2 (a)

ego car pose, ego car rotation, velocity, road information, camera information, and so on.

- `Instance Information`: consists of the instance information dictionary list. The key for each dictionary is the unique ID for the information of one instance in one specific frame. The values include the information: *e.g.*, instance ID (the unique identifier for one instance, *e.g.*, a car, across different frames in one scene), instance global and local translations and rotations, velocity, road information, etc.

The relations between different tables in the database are shown in Fig. 2 (b) (red dashed arrows), *i.e.*, `Scene Information` and `Frame Information`, `Frame Information` and `Ego Information`, `Frame Information` and `Instance Information` are all one-to-many mappings. The detailed fields and their corresponding types for each table are shown in Table 2.

2.2. Task SQLs

As shown in Fig. 2 (b), we define four types of SQL sets (*i.e.*, perception, prediction, risk, and planning with reasoning), which are used for generating different types of instruction-response pairs. Each type of task SQL set consists of several subtask SQLs. For example, the perception SQL set contains six subtask SQLs, *e.g.*, Closest focuses on finding the objects closest to the ego car, and the other subtasks are similar. Note that some high-level SQLs may be inherited from low-level ones following the relational flow of autonomous driving tasks [2], as shown in blue dashed arrows in Fig. 2 (b). For instance, the prediction SQLs are based on the perception ones, the risk SQLs are based on both prediction and perception SQLs, and the planning with reasoning SQL is inherited from prediction and risk ones.

Each subtask SQL consists of a subtask function and an instruction prompt. We illustrate the detailed algorithms for 17 subtask SQLs in Algorithm 1-17. In these algorithms, 'query(T, `*args`)' indicates querying the information from the table T with the arguments `*args`. After obtaining the results from the task SQLs, we transfer them into the language descriptions by template or GPT-4 [4]. Regard the planning with reasoning task as the example, after obtaining the queried results $\{R, s, m\}$, where R, s and m are the risk instance dictionary, the future speed and the future motion of the ego car. Then, the response is formulated as '`There are` R `the ego car. Hence the ego car should be` s `and move to` m`.`'

2.3. Evaluation Details

In this section, we show the computation details for different evaluation metrics, *i.e.*, MAE, accuracy, MAP, and BLEU.

MAE. For tasks measured by MAE, we first use the regular expression to obtain the values from the predicted response, *e.g.*, $\hat{m}$. Then the MAE is computed by $|\hat{m} - m|$, where $|\cdot|$ indicates the absolute value.

Accuracy. There are three kinds of subtasks measured by the accuracy, *i.e.*, Closest, Status, and Same Road. Since the predictions of the Closest subtask are the instance categories, we formulate the Closest subtask to the classification tasks. Similarly, for the Status subtask, there are totally two different statuses (*i.e.*, moving and stationary) for the ego car, while other instances have different statuses for different kinds of objects, *e.g.*, for vehicles, there are moving, stopped and parked; for pedestrians, there are moving, standing and sitting. Hence, the Status subtask can also be regarded as the classification task. Finally, the responses to the Same Road task are 'yes' or 'no', which is a binary classification task.

MAP. We evaluate all subtasks in the risk task by the mean average precision (MAP). Since risk tasks aim to find the objects that may have a risk influence on the ego car driving. Hence, we can transfer them to the object detection tasks, which are generally evaluated by MAP.

Algorithm 1 Distance SQL

Input: Instance information ID: i
Instruction prompt p: What is the distance between `ins` and the ego car?.
I_i = query(T_{ins}, i)
`ins` = < cn, x1, y1, x2, y2 > = I_i[‘**Camera Pos**’]
$(x, y) = I_i$[‘**Local-T**’]
$l = \sqrt{x^2 + y^2}$
return l

Algorithm 2 Closest SQL

Input: Frame ID i
Instruction prompt p: What are the closest objects in `view` of the ego car?;
`view` = {front left, front, front right, back left, back, back right, all }.
F_i = query(T_{frame}, i)
d_{min} = inf; I_{min} = dict()
for v in `view` **do**
  for n in F_i[‘**Instance Information ID List**’] **do**
    I_n = query(T_{ins}, n)
    if v in I_n[‘**Camera Pos**’] or v == all **then**
      $(x, y) = I_n$[‘**Local-T**’]
      $d = \sqrt{x^2 + y^2}$
      if $d_{\text{min}} < d$ **then**
        $I_{\text{min}}[v] = I_n$
      end if
    end if
  end for
end for
return I_{min}

BLEU. BLEU [5] is a classical evaluation metric for caption tasks. In this paper, we use BLEU [5] to evaluate the planning with reasoning task, which is similar to captioning.

3. More Qualitative Examples

We show more visualization examples for all 17 subtasks in Fig. 3 (perception tasks), Fig. 4 (Prediction tasks), Fig. 5-7 (risk tasks) and Fig. 8 (planning with reasoning tasks).

Algorithm 3 Instance Number SQL

Input: Frame ID i
Instruction prompt p: How many `object` in `view` of the ego car?;
`object` = {car, truck, pedestrian, barrier, debris, bicycle, bus, construction, ambulance... }
`view` = {front left, front, front right, back left, back, back right, all }.
F_i = query(T_{frame}, i)
N = dict()
for o in `object` **do**
  for v in `view` **do**
    $N[v][o] = 0$
    for n in F_i[‘**Instance Information ID List**’] **do**
      I_n = query(T_{ins}, n)
      if o == I_n[‘**Category**’] **then**
        if v in I_n[‘**Camera Pos**’] or v == all **then**
          $(x, y) = I_n$[‘**Local-T**’]
          $d = \sqrt{x^2 + y^2}$
          $N[v][o] + +$
        end if
      end if
    end for
  end for
end for
return N

Algorithm 4 Speeds SQL

Input: Instance information ID: i
Instruction prompt p: What is the speeds for `ins`?
I_i = query(T_{ins}, i)
`ins` = < cn, x1, y1, x2, y2 > = I_i[‘**Camera Pos**’]
$v = I_i$[‘**Velocity**’]
return v

Algorithm 5 Status SQL

Input: Instance information ID: i
Instruction prompt p: What is the status for `ins`?
I_i = query(T_{ins}, i)
`ins` = < cn, x1, y1, x2, y2 > = I_i[‘**Camera Pos**’]
$s = I_i$[‘**Attribute**’]
return s

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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020. 1

[2] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, et al. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17853–17862, 2023. 2

[3] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Yu Qiao, and Jifeng Dai. Bevformer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers. In *European conference on computer vision*, pages 1–18. Springer, 2022. 1

[4] OpenAI OpenAI. Gpt-4 technical report. 2023. 2

[5] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002. 3

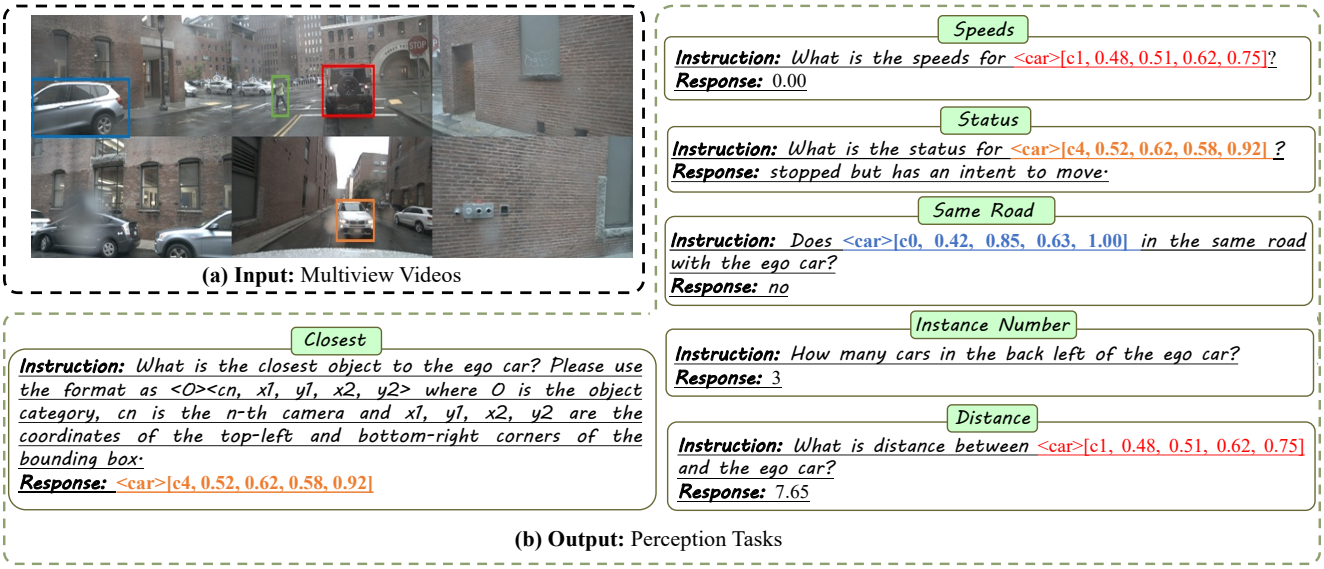


Figure 3. Visualization of our proposed BEV-InMLLM on the perception tasks, which includes six subtasks, *i.e.*, speeds, status, same road, instance number, distance and closest.

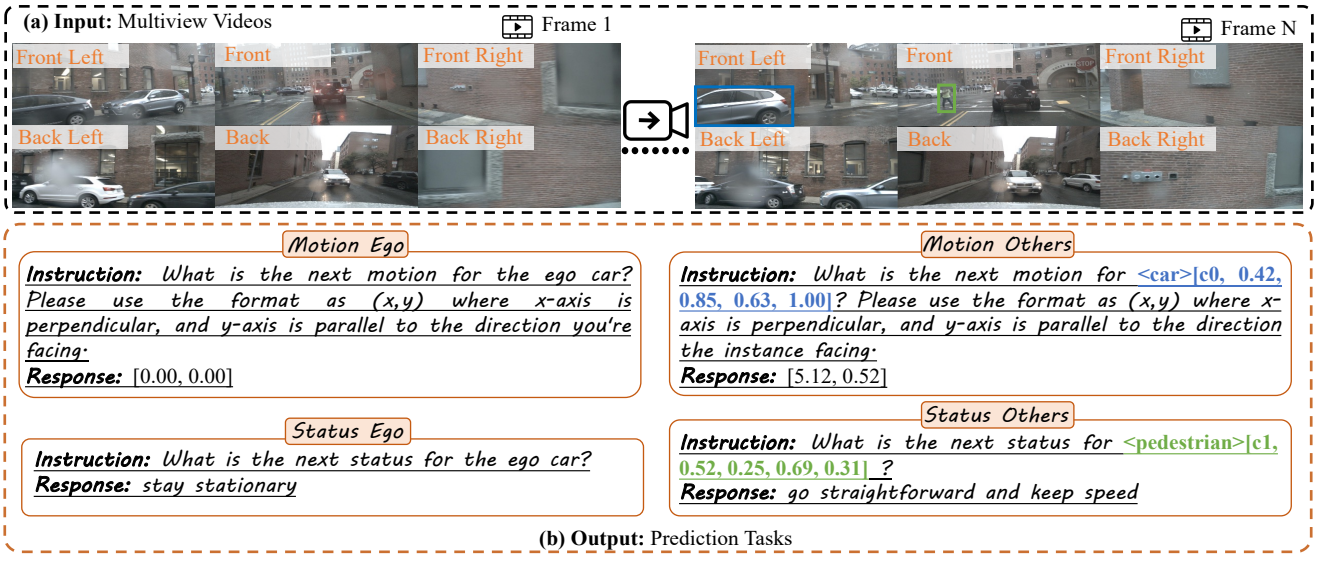


Figure 4. Visualization of our proposed BEV-InMLLM on the prediction tasks, which includes four subtasks, *i.e.*, motion ego, motion others, status ego and status others.

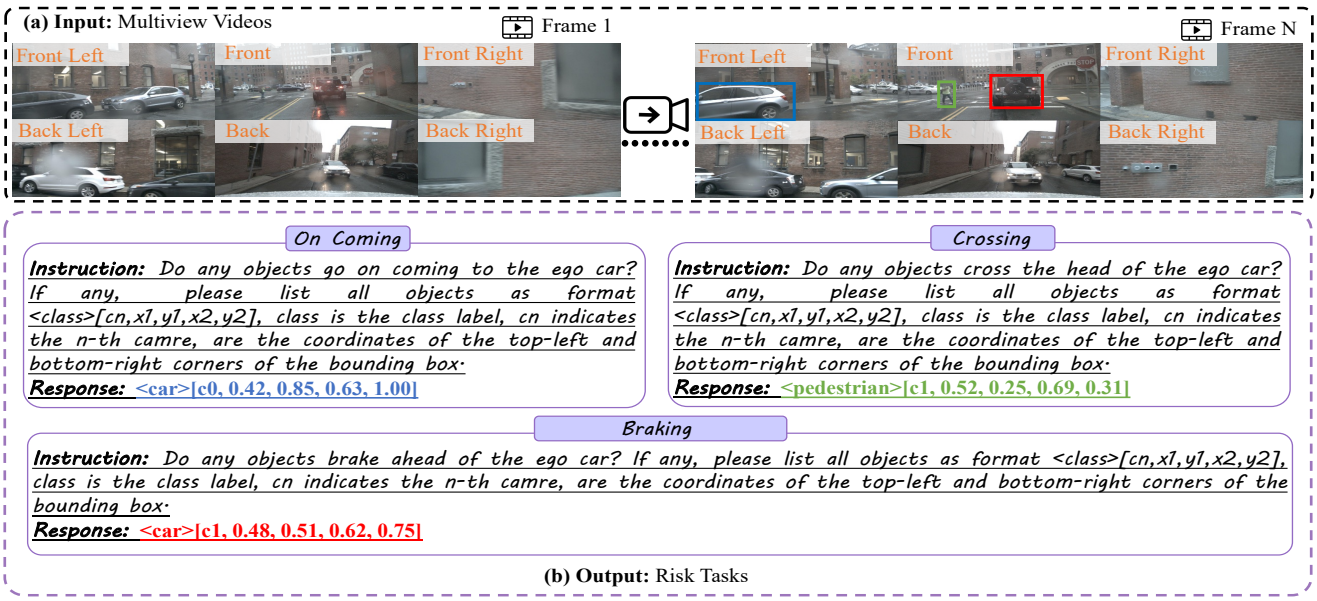


Figure 5. Visualization of our proposed BEV-InMLLM on risk tasks, which includes three subtasks, *i.e.*, on coming, crossing and braking.

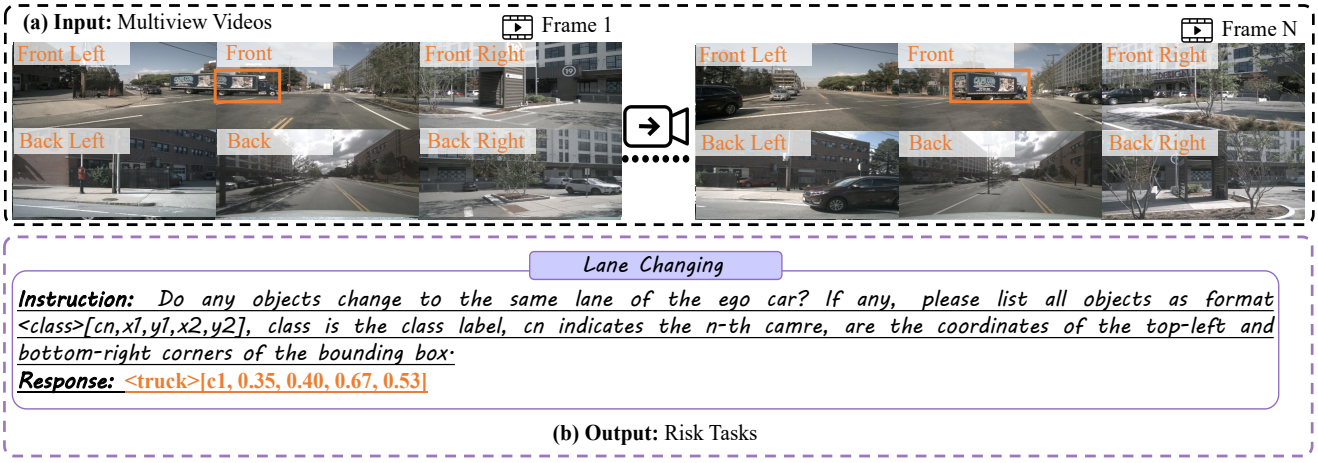


Figure 6. Visualization of our proposed BEV-InMLLM on risk tasks, which includes one subtask, *i.e.*, lane changing.

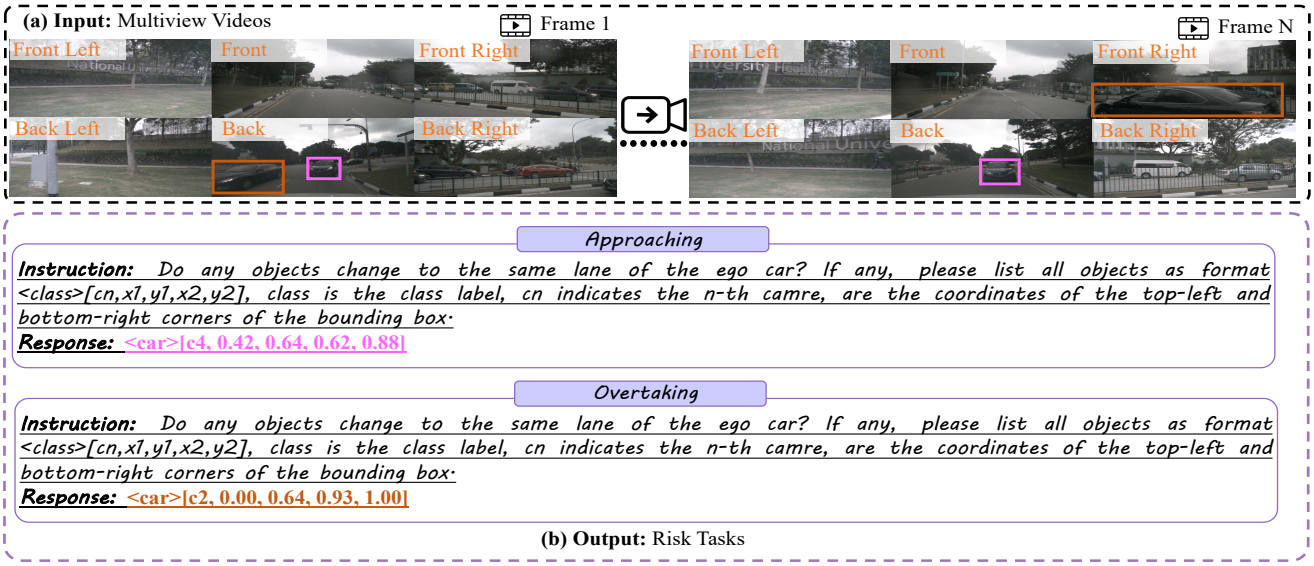


Figure 7. Visualization of our proposed BEV-InMLLM on risk tasks, which includes two subtasks, *i.e.*, approaching and overtaking.

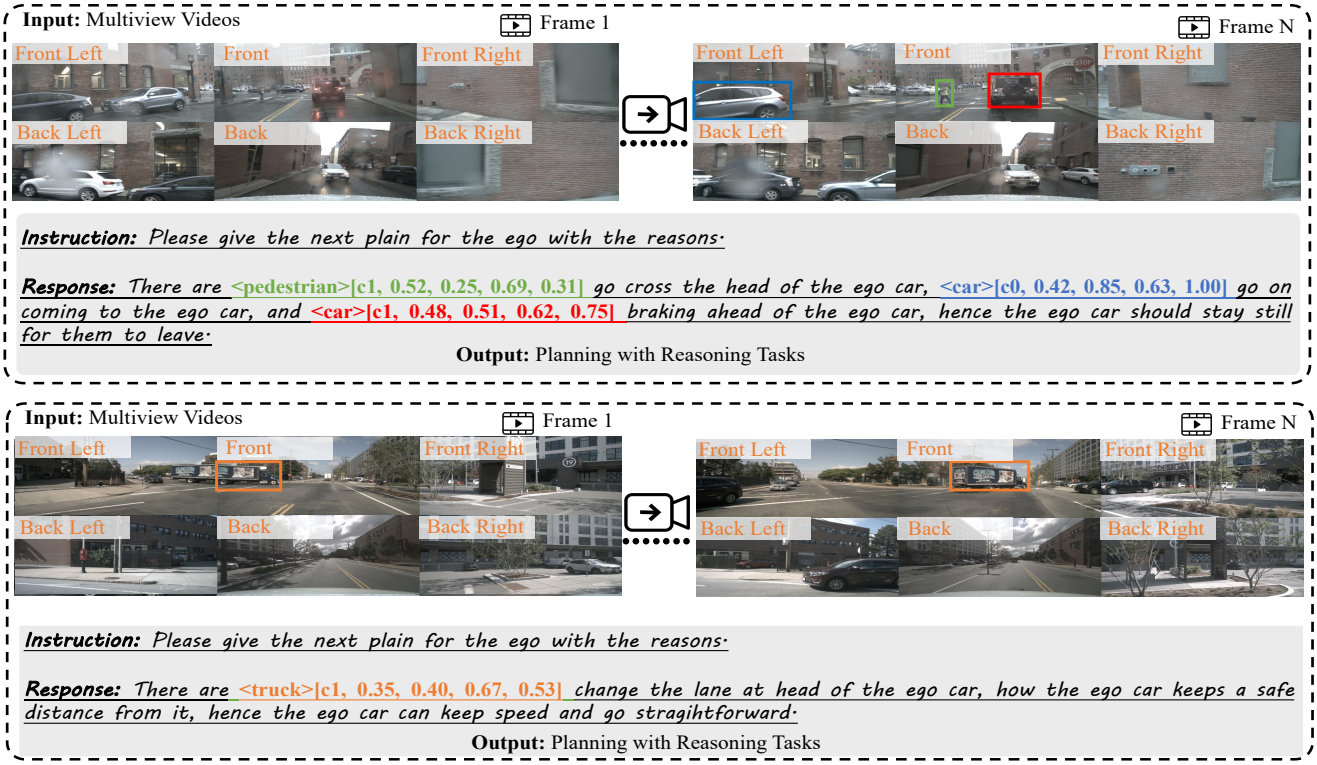


Figure 8. Visualization of our proposed BEV-InMLLM on the planning with reasoning tasks.

Algorithm 6 SameRoad SQL

Input: Instance information ID: i; Frame ID: n
Instruction prompt p: Does `ins` in the same road with the ego car?
I_i = query(T_{ins}, i)
E_n = query(T_{ego}, n)
`ins` = < cn, x1, y1, x2, y2 > = I_i[‘**Camera Pos**’]
r_{ins} = I_i[‘**Road Information**’]
r_{ego} = E_n[‘**Road Information**’]
if r_{ins} ==r_{ego} **then**
  return yes
else
  return no
end if

Algorithm 7 Motion Ego SQL

Input: Current Frame ID: i; Next Frame ID: $i+1$
Instruction prompt p: What is the next motion for the ego car?
E_i = query(T_{ego}, i); p_i = E_i[‘**Pose**’]; r_i = E_i[‘**Rotation**’]
E_{i+1} = query(T_{ego}, $i+1$); p_{i+1} = E_{i+1}[‘**Pose**’]; r_{i+1} = E_{i+1}[‘**Rotation**’]
m = r_i^{-1}.rotate(p_{i+1} - p_i)
return m

Algorithm 8 Motion Others SQL

Input: Current Frame ID: i; Next Frame ID: $i+1$
Instruction prompt p: What is the next motion for `ins`?
F_i = query(T_{frame}, i);
M = dict() ## Motion dictionary for instances
for n in F_i[‘**Instance Information ID List**’] **do**
  I_n = query(T_{ins}, n)
  `ins` = < cn, x1, y1, x2, y2 > = I_n[‘**Camera Pos**’]
  p_n = I_n[‘**Global-T**’]; r_n = I_n[‘**Global-R**’]
  d =
  d = Query(T_{ins}, I_n[‘**Instance ID**’], $i+1$)
  I_d = query(T_{ins}, d)
  p_d = I_d[‘**Global-T**’]; r_d = I_d[‘**Global-R**’]
  $M[n]$ = r_n^{-1}.rotate(p_d - p_n)
end for
return M

Algorithm 9 Status Ego SQL

Input: Current Frame ID: i; Next Frame ID: $i+1$
Instruction prompt: What's the next status for the ego car?
m = Motion Ego (i, $i+1$) ## Algorithm 7
E_i = query(T_{ego}, i); v_i = E_i[‘**Velocity**’]
E_{i+1} = query(T_{ego}, $i+1$); v_{i+1} = E_{i+1}[‘**Velocity**’]
return $\{v_{i+1} - v_i, m\}$

Algorithm 10 Status Others SQL

Input: Current Frame ID: i; Next Frame ID: $i+1$;
Instruction prompt: What's the next status for `ins`?
F_i = query(T_{frame}, i);
M = Motion Others(i,$i+1$) ## Algorithm 8
V = dict() ## Speeds dictionary for instances
for n in F_i[‘**Instance Information ID List**’] **do**
  I_n = query(T_{ins}, n)
  d = I_n[‘**Instance ID**’]
  v_i = Speeds(n); v_{i+1} = Speeds(d) ## Algorithm 4
  $V[n] = v_{i+1} - v_i$
end for
return $\{V, M\}$

Algorithm 11 Overtaking SQL

Input: Previous Frame ID: $i-1$; Current Frame ID: i; Next Frame ID: $i+1$; Threshold distance to the ego car: dis
Instruction prompt: Do any objects overtake the ego car?
F_i = query(T_{frame}, i);
M_{i-1} = Motion Others(i, $i-1$) ## Algorithm 8
M_i = Motion Others(i, $i+1$) ## Algorithm 8
O = list() ## Instance list
for n in F_i[‘**Instance Information ID List**’] **do**
  I_n = query(T_{ins}, n)
  d = Query(T_{ins}, I_n[‘**Instance ID**’], $i-1$)
  v_i = Speeds(n); v_{i-1} = Speeds(d) ## Algorithm 4
  if $M_{i-1}[d][0] < 0$ and $M_{i-1}[d][0] > 0$ and $M_{i-1}[d][1] < dis$ and $M_{i-1}[d][1] < dis$ and $v_i > 0$ and $v_{i-1} > 0$ **then**
    O.append(I_n)
  end if
end for
return O

Algorithm 12 On Coming SQL

Input: Previous Frame ID: $i-1$; Current Frame ID: i; Next Frame ID: $i+1$; Threshold distance to the ego car: dis
Instruction prompt: Do any objects go on coming to the ego car?
F_i = query(T_{frame}, i);
M_{i-1} = Motion Others(i, $i-1$) ## Algorithm 8
M_i = Motion Others(i, $i+1$) ## Algorithm 8
O = list() ## Instance list
for n in F_i[‘**Instance Information ID List**’] **do**
  I_n = query(T_{ins}, n)
  d = Query(T_{ins}, I_n[‘**Instance ID**’], $i-1$)
  v_i = Speeds(n); v_{i-1} = Speeds(d) ## Algorithm 4
  if $M_{i-1}[d][0] > 0$ and $M_{i-1}[d][0] > 0$ and $M_i[d][0] < M_{i-1}[d][0]$ and $M_{i-1}[d][1] < dis$ and $M_i[d][1] < dis$ and $abs(M_i[d][1] - M_{i-1}[d][1]) < dis$)and $v_i > 0$ and $v_{i-1} > 0$ **then**
    O.append(I_n)
  end if
end for
return O

Algorithm 13 Approaching SQL

Input: Previous Frame ID: $i-1$; Current Frame ID: i; Next Frame ID: $i+1$; Threshold distance to the ego car: dis
Instruction prompt: Do any objects approach the ego car?
F_i = query(T_{frame}, i);
M_{i-1} = Motion Others(i, $i-1$) ## Algorithm 8
M_i = Motion Others(i, $i+1$) ## Algorithm 8
O = list() ## Instance list
for n in F_i['**Instance Information ID List**'] **do**
    I_n = query(T_{ins}, n)
    d = Query(T_{ins}, I_n['**Instance ID**'], $i-1$)
    v_i = Speeds(n); v_{i-1} = Speeds(d) ## Algorithm 4
    if $M_i[d][0] < M_{i-1}[d][0]$ and $M_{i-1}[d][1] < dis$ and $M_i[d][1] < dis$ and $abs(M_i[d][1] - M_{i-1}[d][1]) < dis$)and $v_i > 0$ and $v_{i-1} > 0$ **then**
        O.append(I_n)
    end if
end for
return O

Algorithm 14 Crossing SQL

Input: Previous Frame ID: $i-1$; Current Frame ID: i; Next Frame ID: $i+1$; Threshold distance to the ego car: dis; Threshold distance for x direction: dis_x; Threshold distance for y direction: dis_y
Instruction prompt: Do any objects cross the head of the ego car?
F_i = query(T_{frame}, i);
M_{i-1} = Motion Others(i, $i-1$) ## Algorithm 8
M_i = Motion Others(i, $i+1$) ## Algorithm 8
O = list() ## Instance list
for n in F_i['**Instance Information ID List**'] **do**
    I_n = query(T_{ins}, n)
    d = Query(T_{ins}, I_n['**Instance ID**'], $i-1$)
    l_i = Distance(n); l_{i-1} = Distance(d) ## Algorithm 1
    v_i = Speeds(n); v_{i-1} = Speeds(d) ## Algorithm 4
    if $l_i < dis$ and $l_{i-1} < dis$ and $abs(M_i[d][0] - M_{i-1}[d][0]) < dis_x$) and $abs(M_i[d][1] - M_{i-1}[d][1]) > dis_y$) and $v_i > 0$ and $v_{i-1} > 0$ **then**
        O.append(I_n)
    end if
end for
return O

Algorithm 15 Braking SQL

Input: Previous Frame ID: $i-1$; Current Frame ID: i; Next Frame ID: $i+1$; Threshold distance to the ego car: dis; Threshold distance for x direction: dis_x; Threshold distance for y direction: dis_y; Threshold speed: s
Instruction prompt: Do any objects brake ahead of the ego car?
F_i = query(T_{frame}, i);
M_{i-1} = Motion Others(i, $i-1$) ## Algorithm 8
M_i = Motion Others(i, $i+1$) ## Algorithm 8
O = list() ## Instance list
for n in F_i['**Instance Information ID List**'] **do**
    I_n = query(T_{ins}, n)
    d = Query(T_{ins}, I_n['**Instance ID**'], $i-1$)
    l_i = Distance(n); l_{i-1} = Distance(d) ## Algorithm 1
    v_i = Speeds(n); v_{i-1} = Speeds(d) ## Algorithm 4
    if $l_i < l_{i-1} < dis$ and $abs(M_i[d][0] - M_{i-1}[d][0]) > dis_x$) and $M_i[d][1] < dis_y$ and $M_{i-1}[d][1]) < dis_y$) and $v_{i-1} > s$ and $v_i < s$ **then**
        O.append(I_n)
    end if
end for
return O

Algorithm 16 Lane Chaning SQL

Input: Previous Frame ID: $i-1$; Current Frame ID: i; Next Frame ID: $i+1$; Threshold distance to the ego car: dis; Threshold speed: s
Instruction prompt: Do any objects change to the same lane of the ego car?
F_i = query(T_{frame}, i);
M_{i-1} = Motion Others(i, $i-1$) ## Algorithm 8
M_i = Motion Others(i, $i+1$) ## Algorithm 8
O = list() ## Instance list
for n in F_i['**Instance Information ID List**'] **do**
    I_n = query(T_{ins}, n)
    d = Query(T_{ins}, I_n['**Instance ID**'], $i-1$)
    l_i = Distance(n); l_{i-1} = Distance(d) ## Algorithm 1
    v_i = Speeds(n); v_{i-1} = Speeds(d) ## Algorithm 4
    r_i = SameRoad(n); r_{i-1} = SameRoad(d) ## Algorithm 6
    if $l_i < l_{i-1} < dis$ and r_i==yes and r_{i-1}==no and $v_{i-1} > s$ and $v_i > s$ **then**
        O.append(I_n)
    end if
end for
return O

Algorithm 17 Planning with Reasoning SQL

Input: Previous Frame ID: $i-1$; Current Frame ID: i; Next Frame ID: $i+1$; Risk dictionary: `risks` = { 'Overking', 'On Coming', 'Approaching', 'Crossing', 'Braking', 'Lane Changing' }
Instruction prompt: Please give the next plan for the ego car with reasons.
R = dict() ## Risk instance dictionary
m = Motion Ego(i, $i+1$) ## Algorithm 7
s = Status Ego(i, $i+1$) ## Algorithm 9
for `risk` in `risks` **do**
    R[`risk`] = `risk`($i-1$, i, $i+1$) ## Algorithm 11-16
end for
return $\{R, S, M\}$