AoI-centric Task Scheduling for Autonomous Driving Systems

Chengyuan Xu*, Qian Xu†, Jianping Wang*†, and Kui Wu‡, Kejie Lu§, Chunming Qiao¶
*Department of Computer Science, City University of Hong Kong, Hong Kong
†City University of Hong Kong Shenzhen Research Institute
‡Department of Computer Science, University of Victoria, Canada
§ Department of Computer Science, University of Puerto Rico at Mayagüez, Puerto Rico
¶Department of Computer Science, University at Buffalo, USA
chengyuxu2-c,qian.xu@my.cityu.edu.hk, jianwang@cityu.edu.hk, wkui@uvic.ca, kejie.lu@upr.edu, qiao@computer.org

***Abstract*—An Autonomous Driving System (ADS) uses a plethora of sensors and many deep learning based tasks to aid its perception, prediction, motion planning, and vehicle control. To ensure road safety, those tasks should be synchronized and use the latest sensing data, which is challenging since 1) different sensors have different sensing periods, 2) the tasks are inter-dependent, 3) computing resource is limited. This work is the first that uses Age of Information (AoI) as the performance metric for task scheduling in an ADS. We show that minimizing AoI is equivalent to jointly minimizing the response time and maximizing the throughput. We formally formulate the AoI-centric task scheduling problem. To derive practical scheduling solutions, we extend the formulation and formulate the optimal AoI-centric periodic scheduling problem with a given cycle. A reinforcement learning-based solution is designed accordingly. With experiments simulated according to the Apollo driving system, we compare the scheduling performance of the AoI-centric task scheduling with Apollo's schedulers from the perspective of AoI, throughput, and worst case response time. The experiment results show that the maximum AoI in the proposed scheduling solution with 4 cores is lower than that in Apollo's schedulers with 8 cores.**

I. Introduction

A. Background

An autonomous driving system (ADS) consists of many inter-dependent software components to perform various tasks such as perception, motion planning, and control. Fig. 1 shows the algorithm stack of Apollo [1], an open autonomous driving system. In recent years, various models and algorithms have been developed for individual tasks to achieve shorter execution time and higher accuracy. When such tasks are put into the algorithm stack, task scheduling can greatly affect the data freshness in making driving decisions, which essentially affects driving safety.

In a broad sense, scheduling of autonomous driving tasks falls into the category of task scheduling for real-time stream processing, where the inter-dependency of tasks is described by a directed acyclic graph (DAG). Traditional DAG scheduling, i.e., scheduling tasks with precedence constraints on multiple processors, has been studied for years and is recognized as an NP-complete problem [2]–[8]. The performance metrics commonly considered in this area are response time and throughput. Nevertheless, these performance metrics are not sufficient to guarantee driving safety.

The work is supported in part by Hong Kong Research Grant Council under GRF 11218621, and Science and Technology Innovation Committee Foundation of Shenzhen under Grant No. JCYJ20200109143223052.

*Chengyuan Xu and Qian Xu are co-first authors.

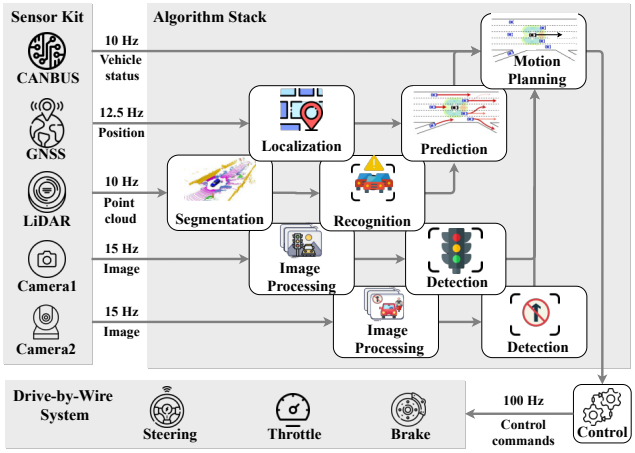


Fig. 1: The autonomous driving System of Apollo.

To ensure driving safety, control commands, such as steering, throttle, and brake commands, should be generated based on the latest environment information. Untimely driving actions may lead to accidents. Thus, task scheduling of ADSs should aim at minimizing the gap between the vehicle's driving decision basis and the real environment. Such a criterion is largely ignored in the literature of autonomous driving scheduling field. This motivates us to introduce a new performance metric for task scheduling in autonomous driving, namely, age of information (AoI) [9], which is defined as the time difference between the time when the control task generates control commands and the minimum timestamp of raw sensor data used in generating control commands. Intuitively, the smaller the AoI, the faster the vehicle reacts to traffic conditions, and the safer the ADS.

B. Motivating Example

To illustrate the criticalness of AoI for driving safety in ADSs, we conduct obstacle avoidance experiments on CommonRoad [10], a collection of benchmarks for motion planning

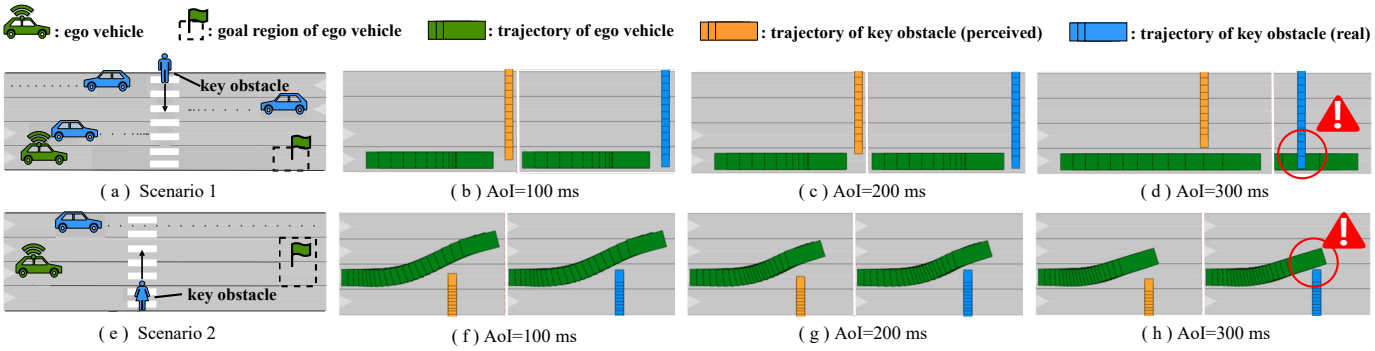


Fig. 2: Simulated driving results of the ego vehicle under three cases with AoI being 100ms, 200ms, and 300ms.

on roads. The scenarios are either transformed from real traffic data (e.g., NGSIM dataset [11]) or manually created based on real road networks. We select two typical scenarios as shown in Fig. 2 (a) and Fig. 2 (e) to illustrate the motion planning results under different levels of AoI. We adopt BMW320i and the Kinematic Single Track Model as the vehicle model, and the A^* algorithm [12] as the motion planner.

We deliberately let the ego vehicle plan its trajectory based on the environmental information collected 100, 200 and 300 ms ago, and then simulate driving according to the planned trajectories in the current environment. In other words, we plan trajectories of the ego vehicle for both scenarios under three cases with AoI being 100, 200 and 300 ms respectively.

- For the case with $AoI = 100$ ms, when the ego vehicle drives according to the trajectory generated by the motion planner, it is still collision-free in both scenarios, as shown in Fig. 2 (b) for scenario 1 and Fig. 2 (f) for scenario 2, since the position of the key obstacle perceived by the ego vehicle is only slightly deviated from the ground-truth.
- For the case with $AoI = 200$ ms (Fig. 2 (c) (g)), though it is collision-free, the distance between the ego vehicle and the key obstacle at the meeting point in both scenarios is much shorter than that in the case of $AoI = 100$ ms.
- For the case with $AoI = 300$ ms, the collision indeed happens as shown in Fig. 2 (d) (h). This is due to the wrongly perceived position of the key obstacle. The ego vehicle even speeds up "before" the key obstacle enters the lane in scenario 1, which may lead to a severe collision in real-world driving.

From the above example, we can conclude that *the deviation of the perception of the environment caused by the high AoI level can easily mislead the autonomous vehicle with an illusion that it is safe, which can cause serious safety hazards and even traffic accidents*. Therefore, limiting and reducing AoI as much as possible should be regarded as one of the key goals of the task scheduling in autonomous driving systems.

C. Challenges and Contributions

While DAG scheduling and AoI minimization are both broadly studied topics, AoI minimization in DAG scheduling for ADSs has rarely been investigated. In particular, we are faced with the following challenges that no existing solution can be directly applied. First, the objective of our scheduling problem is affected by not only the scheduling decisions for various inter-dependent tasks across multiple cores but also the periodically updated sensing data. The asynchronous sensing frequency from multiple sensors on an autonomous vehicle further complicates the problem. Second, from the practical point of view, it is not feasible to optimize and implement the schedule over a very long scheduling horizon. Thus, we need to develop a solution that balances the AoI and computational complexity. Tackling such challenges, this paper makes the following contributions:

- We first introduce age of information (AoI) as a new performance metric for task scheduling in ADSs and show that the objective of minimizing the maximum AoI is equivalent to jointly maximizing the throughput and minimizing the response time, where throughput and response time are often separately considered in conventional task scheduling.
- We formally formulate the AoI-centric task scheduling problem as an ILP problem, referred to as *ILP-AoI*, where the optimal solution in a small scheduling horizon can serve as the lower bound of the large scheduling horizon. We extend the formulation into AoI-centric periodic task scheduling formulation, referred to as *ILP-AoI-II*.
- To efficiently solve the *ILP-AoI-II* problem, we develop a novel *reinforcement learning* (RL) based scheduler. Using *double deep Q network* (Double DQN) to solve the hard combinatorial optimization problem, our RL solution fully leverages the properties derived from *ILP-AoI-II* formulation and the domain knowledge of ADSs to effectively capture the system states and reduce the search space of actions.
- We perform comprehensive evaluation of the AoI-centric task scheduling where the experiments are designed according to Apollo driving system. The results show our scheduling solution with 4 cores outperforms the default schedulers in Apollo with 8 cores.

II. Problem Formulation

In this section, we formulate the AoI-centric task scheduling problem in autonomous driving systems. For ease of reference,

TABLE I: Main Notations

Symbol	Definition
a_p	The latest completion time of tasks in the cross-cycle round on core p
AoI_k	Maximum AoI observed in the k-th execution
b_{ik}, c_{ik}	Starting time and ending time of task i's k-th execution
$e_{ij} \in E$	Directed edge between V_i and V_j
H	Hyper-period of the sensors in ADS
K, K_0	The maximum number of rounds in T and T_0
m, n, P	The number of sensors, tasks, and cores respectively
s_{kz}	Timestamp of sensor z's raw data used in the k-th execution
S_k	Smallest timestamp of raw data used in the k-th execution
T	Scheduling horizon
T_0, T_0'	Cycle length and extended cycle length
$V_i \in V$	The i-th task in task set V
x_{ikpt}	Binary decision about whether task i's k-th execution starts on core p at time t
$y_{kk'z}$	Binary decision about whether task n's k-th execution uses sensor z's k'-th sensing data
τ_z	Sensing period of sensor z
ξ_i	Worst case execution time of task i
μ_k	Binary decision about whether the k-th execution is a dummy round

main notations are summarized in Table I. To save space, we omit the notations related to RL which are easy to follow.

A. ADS Modelling

An ADS mainly consists of three parts, the sensor kit, the algorithm stack, and actuators which are controlled by control commands as shown in Fig. 1. Specifically, the algorithm stack, modelled as a DAG, is represented by a directed graph $G = (V, E)$, where $V = \{V_1, V_2, ..., V_n\}$ is the set of n tasks, and E is the set of direct execution dependency between two tasks. We note that V_n usually represents the motion planning task in an ADS. We assume that there are m sensors in the sensor kit which provide input data periodically to vertices in the DAG.

The control task is usually executed periodically in existing ADSs, e.g., every 10 ms. Considering that the motion planning/replanning task may not be able to generate a new trajectory every 10 ms, the motion planning usually plans the trajectory which can be used for a longer time. Before the motion planning task generates a new trajectory, the control task will be executed multiple times and generate control commands based on the previously generated trajectory

We conduct two one-minute trips of the autonomous vehicle driven by Apollo using its default task scheduler. As shown in Table II, the number of executions of the control task is about 10 times that of the planning task. In the worst case, a trajectory could be used by the control task more than 20 times due to the prolonged time interval between two executions of the motion planning task. From the perspective of driving safety, this should be avoided as much as possible. We aim to tackle this problem in this paper.

B. Definition of AoI in ADSs

Denote c_{nk} as the completion time of the k-th execution of the motion planning task, and $s_{k1}, s_{k2}, \ldots, s_{km}$ as the timestamps of raw data directly or indirectly used by the k-th execution of the planning task from m sensors.

TABLE II: The execution of the planning task vs. the control task in Apollo.

The number of executions		Used times[1]		
Planning	**Control**	**Min.**	**Max.**	**Avg.**
986	9851	6	26	9.9
978	9776	7	23	10.3

[1] The minimum/maximum number of times where a trajectory planned by the planning task is used by the control task.

Let $S_k = \min\{s_{k1}, s_{k2}, \ldots, s_{km}\}$ be the oldest timestamp of the sensor data that the planning task uses in its k-th execution. The control task uses the trajectory planned by the planning task in its k-th execution until the $(k+1)$-th output generated at $c_{n(k+1)}$. Thus, the AoI of the system during $t \in (c_{nk} - c_{n(k+1)}]$ is $t - S_k$. The maximal AoI observed by the control task during the time interval $(c_{nk}, c_{n(k+1)}]$ is $AoI_{(k+1)} = c_{n(k+1)} - S_k$. Note that $AoI_{(k+1)} = c_{n(k+1)} - S_k = (c_{n(k+1)} - c_{nk}) + (c_{nk} - S_k)$ where $c_{n(k+1)} - c_{nk}$ is the time interval between two outputs of the planning task, the reciprocal of the throughput, and $c_{nk} - S_k$ is the response time. Thus, optimizing AoI is equivalent to jointly optimizing throughput and response time in task scheduling of ADSs.

C. Formulation of AoI-centric Task Scheduling in ADSs

In our scheduling problem, we consider a long scheduling horizon, $[0, T]$, in which task n needs to be executed multiple times. We let τ_z be the sensing period of sensor z ($\forall z \in [1, m]$), and without loss of generality, we let τ_1 be the maximum sensing period. Moreover, to simplify the analysis, we assume that all sensors generate the initial sensing data at time 0 and $S_0 = 0$. Assume that the computing unit for autonomous driving has a total of P cores. Let ξ_i be the execution time of task i. Then at most $K = \lfloor \frac{PT}{\sum_{i=1}^{n} \xi_i} \rfloor$ rounds can be executed during T. Due to sensing data availability and dependency in the DAG, it is possible that less than K rounds can be executed during T. For readability, we first formally formulate the problem by assuming that K rounds can be executed. After that, we will present how to add more constraints to the formulation covering the case where less than K rounds can be executed. Next, for S_k with $k \geq 1$, we let $y_{kk'z}$ be a binary decision variable as follows,

$$y_{kk'z} = \begin{cases} 1 & \text{if task } n\text{'s } k\text{-th execution uses sensor } z\text{'s} \\ & k'\text{-th sensing data} \\ 0 & \text{otherwise} \end{cases}$$

where $1 \leq k' \leq K_z = \lfloor \frac{T}{\tau_z} \rfloor$. Since task n uses data from every sensor only once in each execution and it shall use the most recent data from any sensor, we have

$$\sum_{k'=1}^{K_z} y_{kk'z} = 1, \ \forall z \in [1, m], \forall k \in [1, K], \quad \text{and}$$

$$\sum_{k'=1}^{K_z} y_{kk'z} \times k' \leq \sum_{k'=1}^{K_z} y_{k+1,k',z} \times k', \ \forall z \in [1, m], \forall k \in [1, K].$$

Next, by the definition of S_k, we have

$$S_k \leq \sum_{k'=1}^{K_z} [(k'-1) \times \tau_z \times y_{kk'z}], \ \forall z \in [1, m], \forall k \in [1, K].$$

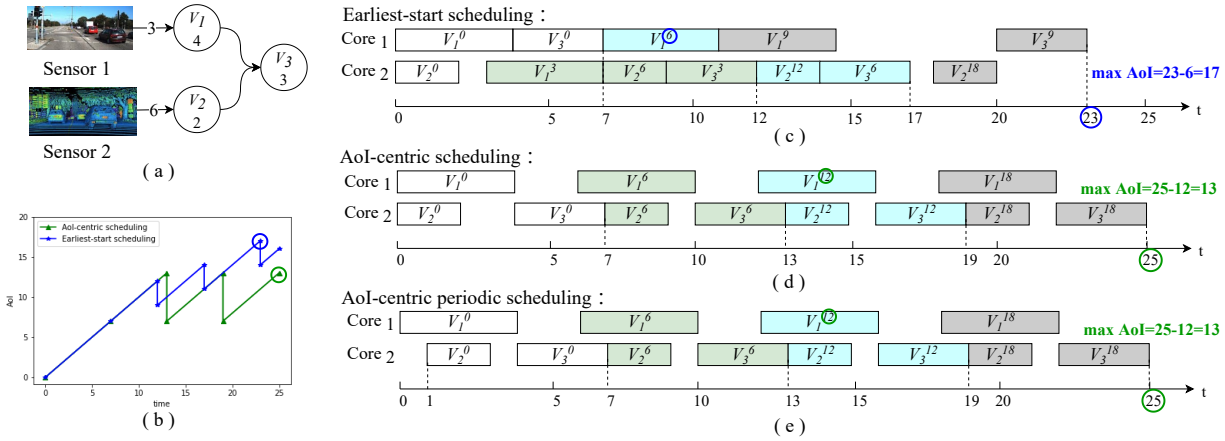


Fig. 3: A demonstration example, where V_i^j means the minimum timestamp of sensing data that V_i uses in its execution is j.

Let b_{ik} be the starting time of task i's k-th execution. If task i takes data of sensor z as one of its inputs directly, we have

$$b_{ik} \geq \sum_{k'=1}^{K_z} [(k'-1) \times \tau_z \times y_{kk'z}], \ \forall k \in [1, K].$$

Let c_{ik} be the completion time of task i's k-th execution, then

$$c_{ik} = b_{ik} + \xi_i \tag{1}$$

Since the starting time of task j cannot be earlier than the completion time of task i if task j depends the output of task i, we have

$$c_{ik} \leq b_{jk}, \text{ if } e_{ij} \in E,$$

Since the completion time of task $i's$ $(k+1)$-th execution cannot be earlier than its k-th execution by, we have

$$c_{ik} \leq c_{i(k+1)}, \ \forall i \in [1, n], \ \forall k \in [1, K-1].$$

We define x_{ikpt} as a binary decision variable as follows.

$$x_{ikpt} = \begin{cases} 1 & \text{if task } i\text{'s } k\text{-th execution starts} \\ & \text{on core } p \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$

Since task i's k-th execution can be performed only once, we have

$$\sum_{p=1}^{P} \sum_{t=0}^{T} x_{ikpt} = 1, \forall i \in [1, n], \forall k \in [1, K] \tag{2}$$

based on which we can further define

$$b_{ik} = \sum_{p=1}^{P} \sum_{t=0}^{T} [t \times x_{ikpt}], \ \forall i \in [1, n], \forall k \in [1, K].$$

To ensure that any core p only executes at most one task at time t, we have

$$\sum_{i=1}^{n} \sum_{k=1}^{K} \sum_{t'=\max(0, t-\xi_i)}^{t} x_{ikpt'} \leq 1, \ \forall p \in [1, P], \forall t \in [0, T].$$

With the aforementioned constraints, we can formulate a problem to minimize the maximum AoI with the objective

$$\textbf{min} \ \max_k (c_{nk} - S_{k-1}). \tag{3}$$

For the case where only less than K rounds are executed during T, we can handle it by allowing a few dummy rounds starting and ending at time 0. We define a binary variable μ_k where $\mu_k = 1$ if the k-th round is actually executed, 0 for otherwise. Then we can modify Constraint (1) to

$$c_{ik} = b_{ik} + \mu_k \xi_i. \tag{1'}$$

Constraint (2) is then modified to

$$\sum_{p=1}^{P} \sum_{t=0}^{T} x_{ikpt} = \mu_k, \forall i \in [1, n], \forall k \in [1, K]. \tag{2'}$$

To enforce that dummy rounds are at the beginning, we can set $\mu_1 \leq \mu_2 \ldots \leq \mu_K = 1$ and $c_{nK} \geq T$.

We refer to the above formulation as *ILP-AoI*. Note that we can obtain the formulation of maximizing the throughput, referred to as *ILP-Throughput*, if we change the objective to

$$\textbf{max} \ \frac{1}{T} \sum_{k=1}^{K} \mu_k. \tag{3'}$$

If we change the objective to

$$\textbf{min} \ \max_k (c_{nk} - S_k) \tag{3"}$$

we can obtain the formulation of minimizing the worst case response time (WCRT), referred to as *ILP-WCRT*. In Section V, we will present the performance of *ILP-AoI*, *ILP-Throughput*, and *ILP-WCRT* to demonstrate the tradeoff among AoI, throughput, and worst case response time.

ILP-AoI defines the general form of the problem. However, the scheduling problem becomes computationally intractable when T is large since the number of variables in the ILP problem is $O(nP^2T^2 + PT^2m)$. A practical solution, from the viewpoints of both computational overhead and real-world implementation, is to derive a periodic scheduling where each period is small. We will formally define and formulate the optimal periodic scheduling problem in the following section.

D. Demonstration Example

We demonstrate different scheduling solutions for a simple system with three tasks as shown in Fig. 3 (a), where V_1 and V_2 are driven by sensors with the sensing period of 3 and 6 respectively, and the execution of V_3 depends on the outputs of V_1 and V_2. P is set to be 2 and T is set to be 25 where 4 rounds are executed. We illustrate the schedule and corresponding system AoI over time under the earliest-start scheduling in which ready tasks are executed as early as

possible, and AoI-centric scheduling formulated in *ILP-AoI*. Fig. 3 (b) depicts the system AoI over time where we can see that the maximum AoI is 17 in earlist-start scheduling and 13 in AoI-centric scheduling. The obtained schedules from earliest-start scheduling and AoI-centric scheduling are illustrated in Fig. 3 (c) and (d) respectively. Furthermore, a AoI-centric periodic scheduling can be obtained as shown in Fig. 3 (e) by delaying the starting time of V_2's first execution by one time slot, which does not change the maximum AoI compared with the solution obtained from *ILP-AoI*.

III. AoI-centric Periodic Task Scheduling

In this section, we first formally define a periodic schedule, then extend the formulation in *ILP-AoI* into the formulation for optimal AoI-centric periodic scheduling.

Definition 1. A task schedule is a *periodic* schedule with cycle length T_0 if it satisfies the following conditions: For any task i, 1) if it starts at time t on core p, then task i starts again at time $t + T_0$ on core p; 2) for every sensor z used by task i, if the timestamp of sensing data from sensor z is s_z when task i starts at time t, then the timestamp of sensing data from sensor z must be $s_z + T_0$ when task i starts at $t + T_0$.

By definition, a periodic schedule can be obtained by repeating a schedule, denoted as π_0, in time intervals $[rT_0, (r+1)T_0]$ for $r \in N$ where each time interval is referred as a cycle. We now define an *optimal* periodic schedule as follows.

Definition 2. A periodic schedule with cycle length T_0 is an *optimal* periodic schedule if the maximum AoI in one cycle with cycle length T_0 is the minimum.

The problem becomes how to determine an appropriate T_0 and construct an optimal schedule π_0 for a given T_0. We now discuss the selection of T_0. According to Definition 1, if the sensing data of a sensor is available at t, it must be available at $t + T_0$ again. Thus, T_0 must be a common multiple of the sensing periods of all sensors. To this end, we define *hyper-period* as follows.

Definition 3. The *hyper-period* denoted as H is the least common multiple of sensing period $\tau_z,\ \forall z \in [1, m]$.

With Definition 1 and Definition 3, T_0 should be qH for $q \in N^+$. Next we discuss the schedule π_0 for one cycle. At this point, one may assume that *ILP-AoI* can be directly applied by setting the time horizon as T_0 and the maximum AoI for π_0 will be the maximum AoI in the entire periodic schedule. However, such a claim is not valid due to two implicit conditions in the formulation.

Firstly, the formulation in Section II implicitly enforces a complete round, i.e., if a task in the DAG finishes its k-th round, all tasks in the DAG shall finish its k-th round. This means that some cores may be idle before T_0 as there is no sufficient time to finish a complete round. The reason behind this is that any task finished after T_0 will not affect the AoI. However, in the periodic schedule, we do need to consider the possibility of having a round in which some tasks start before

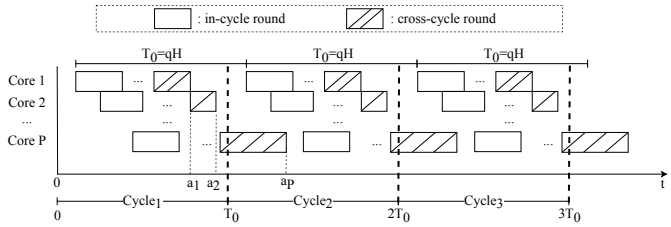


Fig. 4: A periodic schedule with cycle length T_0.

T_0 and task n ends after T_0. We refer to such a round as a cross-cycle round.

Secondly, referring to Eq. 3, we assume that $S_0 = 0$. This is reasonable for a schedule from time 0 because the first output of motion planning is only available at c_{n1}. However, in a periodic schedule, $S_0 \neq 0$ when we consider the AoI of the first round in later cycles. If we directly apply the formulation in *ILP-AoI* to derive the schedule in a cycle T_0 and repeat the obtained schedule to other cycles, the maximum AoI in other cycles may be different from that in $[0, T_0]$.

The above analysis shows the necessity to modify the formulation so that it can be used to construct π_0. Assume that there are K_0 rounds in one cycle. We first address the issue caused by cross-cycle rounds. Our solution is to extend the time horizon for periodic scheduling from $[0, T_0]$ to $[0, T_0 + \sum_i \xi_i]$ so that we can quantify the impact of the cross-cycle round on the AoI of the following cycle. We denote $T_0' = T_0 + \sum_i \xi_i$. For each core p, we further use a_p to represent the latest completion time of tasks in the cross-cycle round on core p. We note that some tasks of a cross-cycle round assigned to a core may finish before T_0, i.e., $a_p \leq T_0$, and some tasks assigned to a core may finish after T_0, i.e., $a_p > T_0$. We have the following constraint:

$$\sum_{t=0}^{T_0'} x_{ikpt}(t + \xi_i) \leq a_p,\ \forall i \leq n, \forall k \leq K_0, \forall p \in [1, P]. \quad (4)$$

Now let us take a look at the cores with $a_p > T_0$. In order to make a periodic schedule, i.e., π_0 for $[0, T_0]$ is also the schedule for $[rT_0, (r+1)T_0]$ for any $r \in N^+$, π_0 should not assign any task to p during the time interval $[0, a_p - T_0]$. Thus, we have the following constraint:

$$a_p - T_0 \leq \sum_{t=0}^{T_0} x_{i1pt} t + (1 - \sum_{t=0}^{T_0} x_{i1pt}) T_0, \forall i \leq n \quad (5)$$

We note that according to Constraint (5), for those cores with $a_p \leq T_0$, tasks can be assigned to them at $t = 0$.

When the first round of the first cycle finishes, the AoI is $AoI_1 = c_{n1} - S_0$. Let us consider the second cycle. The (K_0+1)-th round is the first round in the second cycle. According to Definition 1, we have $c_{n,K_0+1} = c_{n1} + T_0$. When the (K_0+1)-th round finishes, the AoI will be $AoI_{K_0+1} = c_{n,K_0+1} - S_{K_0} = c_{n1} + T_0 - S_{K_0}$. In order to make a periodic schedule, we should have $AoI_1 = AoI_{K_0+1}$, which is $c_{n1} - S_0 = c_{n1} + T_0 - S_{K_0}$. In other words, we should have $S_0 = S_{K_0} - T_0$.

By adding constraints (4) and (5) and setting $S_0 = S_{K_0} - T_0$, we have a new formulation for periodic scheduling, referred to as *ILP-AoI-II*. According to the setting of S_0, we have the following lemma.

Lemma 1. *ILP-AoI-II*, through solving a scheduling problem with K_0 rounds, gives the optimal schedule with K_0+1 rounds where the (K_0+1)-th round is obtained by repeating the 1st round at time T_0.

We use π_0^* to denote a schedule derived based on *ILP-AoI-II* with the time horizon $[0, T_0']$. To build a more intuitive understanding, one periodic schedule with cycle length T_0 is shown in Fig. 4. We have the following lemma.

Lemma 2. By repeating π_0^* at time rT_0 for $r \in N^+$, we can obtain a periodic schedule π with cycle length T_0.

Proof. To prove this lemma, we only need to prove that when we repeat π_0^* at time $0, T_0, 2T_0, \ldots$, we can obtain a periodic schedule, i.e., no two tasks will be assigned to the same core at any time. This is guaranteed by constraints (4) and (5). Consider the r-th cycle started at time $(r-1)T_0$. For each core p, the tasks in the r-th cycle start to occupy core p from time $(r-1)T_0 + max(0, (a_p - T_0))$. According to Constraint (5), tasks in the r-th cycle could not occupy core p until $rT_0 + max(0, (a_p - T_0))$. In other words, time interval $[(r-1)T_0 + max(0, (a_p - T_0)), rT_0 + max(0, (a_p - T_0))]$ on each core p is only used by tasks in the r-th cycle. Thus, no two tasks will be assigned to the same core at any time. Therefore, a periodic schedule with cycle length T_0 can be obtained by repeating π_0^* at time $0, T_0, 2T_0, \ldots$. □

Based on Lemmas 1 and 2, we have the following theorem.

Theorem 1. The periodic schedule π obtained by repeating π_0^* at time rT_0 for $r \in N^+$ is an optimal periodic schedule with cycle length T_0.

IV. RL-based Periodic Scheduling

To solve the periodic scheduling problem effectively, we leverage the recent advance in *reinforcement learning* (RL) for solving hard combinatorial optimization problems, which automates the search of heuristics by training an agent in a self-supervised manner [13]. Here we use the RL agent to replace the ILP solvers and get the periodic schedule in an off-line way, rather than taking it as an online real-time scheduler. That is, the cost of RL-based schedule implementation in ADS can be omitted since it is trained offline.

We model the AoI-centric periodic scheduling problem as a discrete-time Markov decision problem (MDP) and then use Double DQN [14], an RL model suitable for discrete action scenarios, to solve it. We consider the optimal scheduling problem as an MDP with 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ on state space $\mathcal{S}$, action space $\mathcal{A}$, transitional probability function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^+$, and reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. And in our context, the *agent*, i.e., the task scheduler, receives system state $\sigma_l \in \mathcal{S}$ from the environment $\mathcal{E}$ at step l, takes scheduling action $\eta_l \in \mathcal{A}$, and receives reward $r_l \in \mathbb{R}$ at the end of step l. Next, we define the key components in our RL method as follows and then show our training algorithm.

Environment $\mathcal{E}$: To represent the environment for task schedule in $[0, T_0]$, we store the complete information of the DAG and we use a $P \times T_0'$ resource table. For each interaction between the agent and $\mathcal{E}$, we define a decision point (t, p), indicating that core p is available for executing a task at time slot t. Consequently, for step l, $\mathcal{E}$ provides the system state σ_l at decision point (t_l, p_l) to the agent, then updates the resource table according to action η_l, and uses the resource table to find the next decision point (t_{l+1}, p_{l+1}), as shown in Fig. 5.

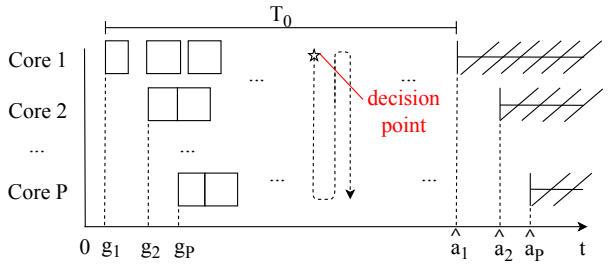


Fig. 5: The environment of RL.

An episode for generating a schedule π_0 with cycle length T_0 is completed when no resource is available on any core in the future time slots. Specifically, assume g_p $(0 \le g_p < T_0)$ is the time of the first execution on core p during the process generating periodic schedule π_0, then the same task will be executed at $g_p + T_0$ on core p when we repeat π_0 in the next cycle. In other words, the resource on core p after time slot $\hat{a}_p = g_p + T_0$ is not available for scheduling in π_0. Since a_p represents the latest completion time of the cross-cycle round on p, $\hat{a}_p$ is an upper limit of a_p to guarantee the periodic constraints in (4) and (5) that we define in Section III. An episodic process is shown in Fig. 5.

State Space $\mathcal{S}$: The state space is designed for providing the complete knowledge required for agent's decision making. To be specific, $\forall \sigma \in \mathcal{S}$, σ=[[sensor related features], [task related features], [core related features]], which is composed of the following three parts:

- Sensor related features: For each sensor z, the features include (1) the sensing period τ_z, (2) the current age of information in terms of sensor z in the previous round and the current round, $t - s_{(k-1)z}$ and $t - s_{kz}$ respectively, (3) the time gap between the current time and the time when sensor z will provide new data $\tau_z - t \bmod \tau_z$.
- Task related features: For each task i, the features include (1) the execution time ξ_i, (2) the time gap between the current time and the time when task i starts in the current round of execution (the value is -1 when this task has not been scheduled in this round), (3) whether task i is ready to be executed.
- Core related features: The features include the index of the currently considered core p which is indicated by the decision point (t, p), and the time gap between the current time and the next available time of each core.

Action Space $\mathcal{A}$: Based on the designed interaction pattern with the environment, the agent only needs to pick one task to execute at each decision point instead of selecting any subset of n tasks that requires 2^n action space. In our RL design, we define the action space as $\eta \in \mathcal{A} = \{0, 1, ..., n\}$, where

$\eta = i,\ i \in [1, n]$ means the agent chooses to schedule task i, and $\eta = 0$ means no task is scheduled at this decision point. With this design, the action space increases linearly with the number of tasks in the DAG. Finally, if task i has a direct data dependency with sensor z and it is scheduled at time slot t, it automatically takes the latest frame of sensor data, with a timestamp of $\lfloor \frac{t}{\tau_z} \rfloor \times \tau_z$.

Action Mask: In the action space $\mathcal{A}$, some scheduling actions may not be feasible or appropriate under a specific state because not all tasks are ready at each step, e.g., dependencies of some tasks are not satisfied yet. Allowing the agent to explore infeasible actions will result in a massive meaningless interaction experience, which hinders the agent from accessing good strategies. Therefore, we provide an action mask at each step to indicate the allowed actions. In our design, both the state and the action mask are provided to the agent at each step, which can significantly reduce the search space and speed up the learning. Moreover, although the ILP solver cannot solve the optimal periodic scheduling directly when T_0 is large, it can still provide knowledge to avoid meaningless actions, which helps simplify the training process of RL. To be specific, the RL agent can take the sensor data selection of the *ILP-AoI-II* solution in a shorter time horizon as a good starting point. If task i uses data from sensor z and $\sum_{k'=1}^{K_z}[(k'-1)\tau_z y_{kk'z}] = s_z$ in the *ILP-AoI-II* solution, then scheduling action of task i's k-th execution in the RL will not be legal in the action mask before s_z accordingly, because there is no reason to choose any older data compared with the existing solution, which serves as an expert experience.

Reward $\mathcal{R}$: For the reward function, the most straightforward design is to return a signal based on the maximum AoI of the generated π_0 when one episode ends. However, this naïve scheme suffers from the sparse reward problem because the reward signals are 0 in all but the last interaction. Therefore, the agent can hardly learn which scheduling actions are good, especially when the cycle length T_0 is large. To solve this problem, we adopt reward shaping as follows. Note that n is the total number of tasks in the DAG, and λ is a scaling parameter.

$$r_l = \begin{cases} \frac{\lambda n}{AoI_k}, & \eta_l = n \\ \frac{\lambda n}{c_{n1}+T_0-S_{K_0}}, & \text{episode terminated} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

In this reward function, when task n is scheduled at step l, meaning that one round of execution from task 1 to task n is just settled, the agent will get a reward according to the AoI of this round, and when one episode ends, a periodic schedule π_0 is settled so that a final reward is given according to the AoI of the first and the cross-cycle rounds, i.e., $AoI_{K_0+1} = c_{n,K_0+1} - S_{K_0}$=$c_{n1} + T_0 - S_{K_0}$. In both of these two cases, we design the reward inversely proportional to the objective so that the agent can be guided to learn how to minimize the AoI when generating a periodic π_0.

Training Algorithm: Based on the definitions above, we design Algorithm 1 to train our RL-based scheduler. In this algorithm, lines 7-17 facilitate experience replay to accelerate the training of the RL model. Following the design of Double DQN in [14], we use the trained RL network with parameter θ to find a desired action (line 13), and then use a target network with parameter θ' to evaluate the expected reward of this action (line 14). According to [14], such an approach can address the overoptimism problem associated with a commonly used DQN design, in which the target network is used to find an action and estimate the reward. Finally, in our algorithm, the action mask is applied in line 5 to explore a valid action.

Algorithm 1 The Procedure to Train A Double DQN Agent

Input: **Scheduling parameters:** DAG, P, T_0, and a reference solution obtained by solving *ILP-AoI-II* in a small time period
Training parameters: replay buffer capacity N_r, number of episodes M, exploration factor ϵ, training batch size N_b, discount factor γ, target update frequency N^-
Output: Periodic schedule π_0

1: Initialize replay buffer $\mathcal{D}$, Q network parameters θ, target network parameters θ^-
2: **for** episode $e \in \{1, 2, ..., M\}$ **do**
3: Initialize environment $\mathcal{E}$
4: **for** $l \in \{0, 1, ...\}$ **do**
5: Select a valid action η_l by the ϵ-greedy policy
6: Receive reward r_l and next state σ_{l+1}
7: Add $(\sigma_l, \eta_l, r_l, \sigma_{l+1})$ to $\mathcal{D}$, drop oldest tuple if $|\mathcal{D}| > N_r$
8: Uniformly sample N_b tuples $(\sigma, \eta, r, \sigma') \in \mathcal{D}$
9: **for** each sample $(\sigma, \eta, r, \sigma')$ **do**
10: **if** σ' is terminal **then**
11: $Y = r$
12: **else**
13: $\eta_{max} = \text{argmax}_{\eta'} Q(\sigma', \eta'; \theta)$
14: $Y = r + \gamma Q(\sigma', \eta_{max}; \theta^-)$
15: **end if**
16: Update θ using gradient of loss $||Y - Q(\sigma, \eta; \theta)||^2$
17: **end for**
18: Replace target parameters $\theta^- \leftarrow \theta$ every N^- steps
19: **end for**
20: **end for**

V. Performance Evaluation

We conduct experiments based on the settings of the Apollo ADS and evaluate the performance of different schedulers from the following perspectives. Firstly, given a small T, we present the optimal scheduling of *ILP-AoI*, *ILP-Throughput*, and *ILP-WCRT* to demonstrate the tradeoff among AoI, throughput, and WCRT. Secondly, we compare the performance of the proposed RL-based scheduling with the optimal *periodic* solution obtained from *ILP-AoI-II*, and optimal solution obtained from *ILP-AoI* for a given small T_0 to demonstrate the effectiveness of the proposed RL solution. Thirdly, we compare the performance of the proposed RL-based scheduling with two event-driven scheduling approaches in the Apollo system for a large T.

A. Workloads

The experiments are set according to the workload in Apollo, such as the algorithm stack, the sensing frequency, and execution time of each task. Log files are first collected by a co-simulation experiment using Apollo and LGSVL simulator [15] on a device officially recommended with Intel® Core™ i7-9700K 8-core CPU, 64 GB RAM, and GeForce

Task Name	WCET (ms)
V_1: Localization	18.2
V_2: Segmentation	49.8
V_3: Image Processing	26.3
V_4: Image Processing*	24.9
V_5: Recognition	8.4
V_6: Traffic Light	48.4
V_7: Traffic Light*	39.2
V_8: Prediction	18.6
V_9: Planning	86.4

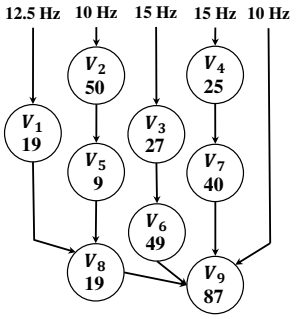
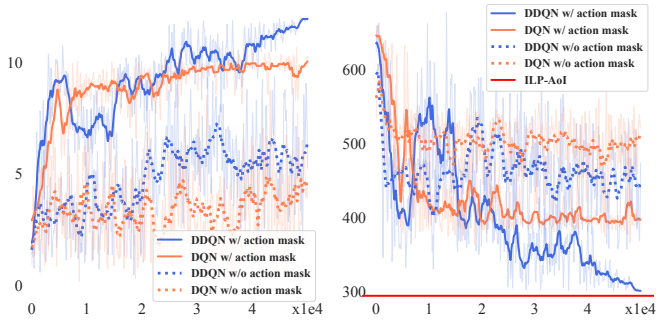


Fig. 6: WCRT for tasks in Apollo's algorithm stack and the constructed DAG for evaluation.



(a) Accumulated reward vs. Steps (b) Maximum AoI vs. Steps

Fig. 7: Training of an RL agent with $T_0 = 2H$, $P = 4$.

RTX 2080 Ti Graphics Card. The ego vehicle is driven by Apollo from an initial position to the destination in the scenario simulated by LGSVL. Fig. 6 shows the *worst case execution time* (WCET) of different tasks and the DAG for evaluation experiments which is constructed accordingly. There are five sensors with sensing frequency 12.5, 10, 15, 15 and 10 Hz respectively, based on which the *hyper-period* of the DAG is $H = 400$ ms. The default resource configuration on the Apollo ADS is 8 CPU cores.

B. Baselines

The baselines are the classic scheduling and choreography scheduling proposed in Apollo Cyber RT [16]. Specifically, the classic scheduling divides the algorithm stack into two groups, $\{V_1, V_2, V_5, V_8, V_9\}$ and $\{V_3, V_4, V_6, V_7\}$, and assigns each group to four cores separately. Priorities of tasks in the first group increase sequentially, and the task with the highest priority in the ready task pool is always executed first. Tasks in the second group have the same priorities and the task with the earliest ready time is executed first. For the choreography scheduling, $\{V_3, V_4, V_6, V_7\}$ are assigned to three cores with equal priorities, and the other tasks are bound to the remaining cores one-to-one.

C. Implementation of the RL Agent

The setting of the RL agent in Algorithm 1 is as follows. The RL agent uses a 3 hidden layer neural network with each width of 128 to approximate the Q-value and uses Adam optimizer with learning rate 10^{-5}. The hyper parameters include scale parameter $\lambda = 20$, exploration factor $\epsilon = 0.1$, replay buffer capacity $N_r = 20000$, training batch $N_b = 64$,

TABLE III: Comparison of three ILP solvers.

Settings	Solver	Throughput (rounds/s)	WCRT (ms)	Max AoI (ms)
$P = 3$ $T = 2H$	*ILP-Throughput*	**9.3**	333	400
	ILP-WCRT	8.4	**180**	398
	ILP-AoI	8.9	191	**320**
$P = 4$ $T = 2H$	*ILP-Throughput*	**10**	286	386
	ILP-WCRT	8.4	**175**	396
	ILP-AoI	**10**	**175**	**294**

TABLE IV: Comparison of RL and ILP solvers.

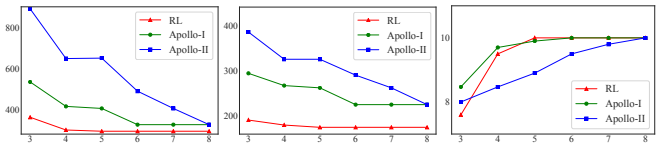
Max AoI (ms) T_0 / Solvers	P=3					P=4				
	H	2H	3H	4H	5H	H	2H	3H	4H	5H
ILP-AoI	300	320	⟋	⟋	⟋	294	294	⟋	⟋	⟋
ILP-AoI-II	362	362	⟋	⟋	⟋	362	294	⟋	⟋	⟋
RL	362	362	362	342	342	362	300	294	294	294

and update frequency $N^- = 500$. We conduct our algorithm with the same network structure under different settings of P where $3 \leq P \leq 8$. Fig. 7 shows an instance of the training process. The blue solid curves in Fig. 7(a) and Fig. 7(b) depict the accumulated reward and the maximum AoI observed in each episode during training the Double DQN model with $P = 4$ and $T_0 = 2H$. We also explore the performance of the DQN model for the same setting, and the performance of both Double DQN and DQN models without adopting action mask. The result shows that the agent learns better strategies more steadily and quickly using Double DQN with action mask, which reaches nearly optimal at around 50000 steps.

D. Experimental Results

To demonstrate the tradeoff among three different performance metrics, i.e., AoI, throughput, and WCRT, we first compare the optimal scheduling obtained by *ILP-AoI*, *ILP-Throughput*, and *ILP-WCRT*. We conduct two experiments with setting $P = 3$, $T = 2H$ and $P = 4$, $T = 2H$ respectively. Results in Table III show that, when throughput and WCRT are optimized separately by *ILP-Throughput* and *ILP-WCRT*, the performance in the other two performance metrics is poor, while for the schedule in *ILP-AoI*, both the throughput and WCRT are nearly optimal when $P = 3$. When $P = 4$, the solution obtained through *ILP-AoI* achieves the optimal solution in all three performance metrics. This confirms our analysis in Section II that optimizing AoI is equivalent to jointly optimizing throughput and response time.

To demonstrate the performance of the RL-based AoI-centric scheduling, we compare it with the schedule obtained by *ILP-AoI* and the schedule obtained by *ILP-AoI-II* under $P = 3$ and $P = 4$. The scheduling solutions for $T_0 = H$ and $T_0 = 2H$ are derived for *ILP-AoI*, *ILP-AoI-II*, and RL. For $T_0 > 2H$, the problem size is too large for *ILP-AoI* and *ILP-AoI-II*, while our RL-based method can still solve the problem. We note that the solution obtained by *ILP-AoI* when $T = 2H$ is a lower bound of the problem for $T > 2H$. Thus, we compare the RL-based solution with the optimal periodic solution obtained from *ILP-AoI-II* for $T_0 = H$ and $T_0 = 2H$. When $T_0 > 2H$, we compare the RL-based solution with the lower bound obtained by *ILP-AoI* when $T_0 = 2H$. As



(a) Max. AoI vs. P (b) WCRT vs. P (c) Throughput vs. P

Fig. 8: Compare RL when $T_0 = 2H$ with Apollo's baseline schedulers: classic (Apollo-I) and choreography (Apollo-II).

TABLE V: The number of task executions.

Schedule	Configuration	The number of task executions
Apollo-I	$P = 4$	V_1:1250, V_6:1369, V_9:999 $V_2V_5V_8$:1000,$V_3V_4V_7$:1493
Apollo-II	$P = 4$	V_1:1215, V_2:1002, V_3:1363 V_4:879, V_5:967, V_6:743 V_7:712, V_8:967, V_9:965
RL	$P = 4,\ T_0 = 2H$	V_1V_2:999, others:998

shown in Table IV, when $P = 3$, the maximum AoI achieved in the RL-based solution is the same to that in *ILP-AoI-II* for $T_0 = 2H$. The maximum AoI achieved in the RL-based solution for $T_0 = 5H$ is very close to the lower bound of the problem. When $P = 4$, the RL-based solution for $T_0 > 2H$ can achieve the lower bound of the problem. This demonstrates the effectiveness of the RL-based method.

We compare our RL-based AoI-centric task scheduling with two baselines in Apollo. We set the schedule configuration according to the same principle in Apollo for $3 \leq P \leq 8$. Specifically, for the classic schedule, cores are allocated evenly to two task groups. When the number of cores is odd, the group with more tasks will get one more core. For the choreography schedule, starting from 7, every time when P is decreased by one, the task with the smallest index is unbound from a core. As shown in Fig. 8 (a), the AoI performance of the RL-based AoI-centric scheduling with 4 cores is even better than the two baselines with 8 cores. Fig. 8 (b) and (c) show that the throughput and WCRT in RL-based AoI-centric scheduling are better than that in two baselines for $P \geq 5$.

To have a closer look at the difference between baselines and our AoI-centric schedule, Table V shows the number of task executions during 100 seconds. Since the slowest sensing frequency is 10 Hz, the motion planning task can be executed at most 1000 times in 100 seconds. Our experiments observe that the number of executions for the motion planning task in our method and two baselines is 998, 999, and 965 respectively. However, compared with our RL-based scheduling, many unnecessary executions are performed in Apollo's default schedulers. For example, V_1, V_3, V_4, V_6, V_7 are executed more often than necessary in Apollo-I, which leads to a waste of computing resource. moreover, serious resource competition may happen under some scheduling configuration, for example, the number of task executions in Apollo-II is unbalance which leads to a poor performance on AoI.

VI. Related Work

In this section, we discuss the most relevant works in real-time task scheduling from the perspectives of DAG-aware scheduling and AoI-based scheduling.

The real-time scheduling of inter-dependent tasks represented as a DAG with precedence constraints on multiple processors has been studied for years, with general objectives of minimizing the makespan of the schedule or meeting the deadline of job completion time [3], [4]. A branch scheduling is proposed in [17] which improves the classical critical path method for DAG scheduling in modern distributed systems. Considering both task dependencies and heterogeneous resource demands at the same time, [18] proposes a scheduler based on DRL and Monte Carlo Tree Search for complex jobs. In autonomous driving systems, the work in [19] designs the scheduler using Kalray MPPA-256 as the many-core processor with consideration of asynchronous sensing periods among sensors. A heuristic scheduling algorithm is designed in [20] based on list scheduling to process streamed data from onboard and external sensors such as V2V efficiently.

AoI-based scheduling generally aims to minimize the average and peak AoI in queueing systems. The scheduling policies proposed in the existing works are constraint-specific with different considerations, e.g., active and buffered sources in source-destination communication links [21], deadline constrained real-time traffic in an ad hoc wireless network [22]. The work in [23] considers the random status updates arrival at a source node while the work in [24] takes AoI requirement at each source node into consideration. Considering the general and heterogeneous sampling behaviors, varying sample size, and multiple data transmission units, the work in [25] develops a near-optimal low-complexity scheduling algorithm. Though AoI has been proposed in vehicular networks [26] to avoid collisions, no existing work has applied AoI-based scheduling in autonomous driving systems due to the challenge of inter-dependent tasks.

VII. Conclusion

In this paper, we proposed an AoI-centric task scheduling for autonomous driving systems to minimize the maximum age of information of the algorithm stack represented by a DAG. We first introduced the AoI as a performance metric for task scheduling in autonomous driving, and formulated the AoI-centric scheduling problem as an ILP. Considering that it is hard to predict the length of a scheduling horizon, we extended the formulation and re-formulated the optimal AoI-centric *periodic* scheduling problem. With the knowledge from the AoI-centric periodic scheduling, we developed an RL-based periodic AoI-centric scheduling. Experiments designed according to the Apollo driving system show that the RL-based AoI-centric periodic scheduling can achieve near-optimal solution and outperform the default schedulers in Apollo from the perspectives of AoI, throughput, and the worst case response time. In the future work, we will take the tasks' execution time dynamics into consideration and study the driving scenario specific schedule with low AoI and high utilization of the cores.

References

[1] Baidu Apollo, "An open autonomous driving platform," 2019.

[2] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. D. Gill, "Parallel real-time scheduling of dags," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3242–3252, 2014.

[3] S. Baruah, "Improved Multiprocessor Global Schedulability Analysis of Sporadic DAG Task Systems," in *Proc. of IEEE 26th Euromicro conference on real-time systems*, 2014, pp. 97–105.

[4] M. A. Serrano, A. Melani, M. Bertogna, and E. Quiñones, "Response-time Analysis of DAG Tasks under Fixed Priority Scheduling with Limited Preemptions," in *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016.

[5] J. Fonseca, G. Nelissen, V. Nelis, and L. M. Pinho, "Response Time Analysis of Sporadic DAG Tasks under Partitioned Scheduling," in *Proc. of IEEE 11th Symposium on Industrial Embedded Systems (SIES)*, 2016.

[6] S. Baruah, "The Federated Scheduling of Systems of Mixed-Criticality Sporadic DAG Tasks," in *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, 2016.

[7] R. Sakellariou and H. Zhao, "A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems," in *Proc. of IEEE 18th International Parallel and Distributed Processing Symposium*, 2004.

[8] M. A. Serrano and E. Quinones, "Response-time Analysis of DAG Tasks Supporting Heterogeneous Computing," in *Proc. of the 55th Annual Design Automation Conference*, 2018.

[9] S. Kaul, M. Gruteser, V. Rai, and J. Kenney, "Minimizing Age of Information in Vehicular Networks," in *Proc. of IEEE 8th Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, 2011.

[10] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable Benchmarks for Motion Planning on Roads," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017.

[11] FHWA, U.S. Department of Transportation, "NGSIM: Next Generation Simulation." http://www.fhwa.dot.gov/, 2007.

[12] J. Yao, C. Lin, X. Xie, A. J. Wang, and C. Hung, "Path Planning for Virtual Human Motion Using Improved A* Star Algorithm," in *7th International Conference on Information Technology: New Generations*, 2010.

[13] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement Learning for Combinatorial Optimization: A Survey," *Computers & Operations Research*, p. 105400, 2021.

[14] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[22] N. Lu, B. Ji, and B. Li, "Age-based Scheduling: Improving Data Freshness for Wireless Real-Time Traffic," in *Proc. of the 18th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2018.

[15] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, and S. Kim, "LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving," in *Proc. of IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020.

[16] Apollo CyberRT, https://github.com/ApolloAuto/apollo/tree/master/cyber, 2019.

[17] Z. Hu, D. Li, Y. Zhang, D. Guo, and Z. Li, "Branch Scheduling: DAG-Aware Scheduling for Speeding up Data-Parallel Jobs," in *Proc. of IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*.

[18] Z. Hu, J. Tu, and B. Li, "Spear: Optimized Dependency-aware Task Scheduling with Deep Reinforcement Learning," in *Proc. of IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019.

[19] S. Igarashi, Y. Kitagawa, T. Ishigooka, T. Horiguchi, and T. Azumi, "Multi-rate DAG Scheduling Considering Communication Contention for NoC-based Embedded Many-core Processor," in *Proc. of IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2019.

[20] J. Rho, T. Azumi, M. Nakagawa, K. Sato, and N. Nishio, "Scheduling Parallel and Distributed Processing for Automotive Data Stream Management System," *Journal of Parallel and Distributed Computing*, vol. 109, pp. 286 – 300, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731517302010

[21] R. Talak, S. Karaman, and E. Modiano, "Optimizing Information Freshness in Wireless Networks under General Interference Constraints," *IEEE/ACM Transactions on Networking*, vol. 28, no. 1, pp. 15–28, 2019.

[23] M. Costa, M. Codreanu, and A. Ephremides, "Age of Information with Packet Management," in *Proc. of IEEE International Symposium on Information Theory*, 2014.

[24] C. Li, S. Li, Y. Chen, Y. Thomas Hou, and W. Lou, "AoI Scheduling with Maximum Thresholds," in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2020.

[25] C. Li, S. Li, and Y. T. Hou, "A general model for minimizing age of information at network edge," in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2019.

[26] S. Kaul, R. Yates, and M. Gruteser, "Real-time status: How often should one update?" in *Proc. of IEEE Conference on Computer Communicatios (INFOCOM)*, 2012.