Modeling and Scheduling of Fusion Patterns in Autonomous Driving Systems

Hoora Sobhani[0009−0009−6611−9374] and Hyoseung Kim[0000−0002−8553−732X]

University of California, Riverside (UCR)
{hsobh002, hyoseung}@ucr.edu

Abstract. In Autonomous Driving Systems (ADS), Directed Acyclic Graphs (DAGs) are widely used to model complex data dependencies and inter-task communication. However, existing DAG scheduling approaches oversimplify data fusion tasks by assuming fixed triggering mechanisms, failing to capture the diverse fusion patterns found in real-world ADS software stacks. In this paper, we propose a systematic framework for analyzing various fusion patterns and their performance implications in ADS. Our framework models three distinct fusion task types: timer-triggered, wait-for-all, and immediate fusion, which comprehensively represent real-world fusion behaviors. Our Integer Linear Programming (ILP)-based approach enables an optimization of multiple real-time performance metrics, including reaction time, time disparity, age of information, and response time, while generating deterministic offline schedules directly applicable to real platforms. Evaluation using real-world ADS case studies, Raspberry Pi implementation, and randomly generated DAGs demonstrates that our framework handles diverse fusion patterns beyond the scope of existing work, and achieves substantial performance improvements in comparable scenarios.

1 Introduction

Research in Autonomous Driving Systems (ADS) has attracted strong interest from academia and industry, aiming to develop safe and reliable autonomous vehicles. The ADS software stack consists of tightly interconnected components with intricate data dependencies, interacting through a deep processing pipeline [14, 19, 39]. These components include sensors, algorithms, and actuators, all working together under strict timing constraints to ensure safety. They are often represented as nodes in Directed Acyclic Graphs (DAGs), capturing the system's workflow, task dependencies, and inter-task communication.

DAGs have been instrumental in studying task scheduling and resource allocation in ADS, especially to improve key performance metrics like end-to-end latency, sensor time disparity, data freshness, and responsiveness in safety-critical, real-time environments. However, existing DAG scheduling models often fall short in fully capturing the diverse behaviors of real-world ADS, such as Autoware [2]. One major challenge is accurately modeling data *fusion nodes*, which aggregate multiple inputs to generate outputs. In ADS, numerous sensors operate at different sampling rates, passing their data through these fusion nodes. Dealing with the varying sampling rates of sensors is inevitable due to the limited flexibility of sensor hardware (e.g., lidar and camera cannot choose arbitrary sampling periods). Enforcing rate uniformity may cause all tasks to run with the shortest period or with the GCD of all periods. In addition, the fusion nodes add complexity as they can be triggered either by their own timers or by external events with unpredictable arrival times. This complexity deepens in event-triggered cases, where any of the inputs to a fusion node can independently initiate its activation. Such variability in activation timing and patterns complicates modeling and scheduling, as traditional DAG models often assume more uniform or simpler task activation mechanisms. For instance, authors in [32, 33, 37] assumed that a fusion node is triggered by a single dominant input edge, while authors in [25] assumed all nodes, including fusion nodes, are triggered solely by timers. While these models may fit certain applications, existing work focuses on only one type of fusion behavior, overlooking the various types found in real-world ADS such as Autoware.

Beyond fusion node limitations, existing studies typically optimize individual chains, rather than considering the entire DAG holistically with multiple metrics [25, 32, 36]. Even studies that include both timer- and event-triggered tasks [11, 31, 32] fail to capture real-world ADS configurations like branch-then-fusion paths (where multiple outputs fork from a common node and later converge into a single node) and diverse triggering mechanisms. Addressing these gaps is essential to reflect the dynamic and interconnected nature of ADS workflows.

In this paper, we offer a systematic framework for analyzing various fusion patterns in ADS and their impact on real-time performance. To explore the best achievable performance under different fusion strategies, we formulate the DAG scheduling problem and diverse fusion patterns as Integer Linear Programming (ILP) constraints, as ILP allows us to obtain optimal schedules while offering extensibility in constraint modeling. Our approach enables a quantitative comparison of how different fusion strategies affect key performance metrics such as worst-case response time (WCRT), maximum reaction time (MRT), maximum time disparity (MTD), and peak age of information (PAoI), while also uncovering trade-offs of different fusion nodes in ADS design. The optimal schedule generated by our framework can be directly applicable to real-world ADS with static non-preemptive scheduling, such as NVIDIA's STM [7]. We evaluate our framework through comparison against state-of-the-art methods, implementation on a Raspberry Pi, and experiments with randomly generated complex DAGs. Experimental results show that our framework successfully handles various fusion patterns beyond the scope of prior work and achieves substantial improvements over existing methods in comparable scenarios.

2 Related Work

Prior work on DAG scheduling in real-time systems, particularly for ADS applications, can be categorized by the type of chains used to construct DAG models, which differ in data communication and task-triggering mechanisms.

Generally, a chain is a sequence of tasks, where each task depends on data from its predecessor. Tasks within a chain can be triggered by either *timers* or *events*. A widely adopted model is the **cause-effect chain**, traditionally representing multi-rate real-time tasks triggered solely by timers at different rates [3, 29, 34]. These chains have evolved to include event-triggered tasks, enabling responses to specific events rather than relying entirely on timers [28, 30, 31, 33, 37]. We refer to the former as **multi-rate cause-effect chains** and the latter as **enhanced cause-effect chains** throughout this section.

Many studies on cause-effect chains aim to minimize metrics like maximum reaction time (MRT) and maximum data age (MDA), which are typically used to evaluate system responsiveness and data freshness, respectively. For example, Becker et al. eliminated data paths in a DAG that exceed age constraints to improve MRT and MDA [3, 4]. A common goal across these studies is to model task communication across the DAG and explore solutions that streamline data flow and fusion for faster responsiveness to sensor data and improved data freshness. For multi-rate cause-effect chains, Tang et al. [29] compared communication paradigms (implicit, LET, DBP), while Maia et al. [18] explored strategies to adjust communication intervals. Saidi et al. [20] introduced a graph transformation approach to establish execution order, and Verucchi et al. [34] presented a method for converting multi-rate DAG chains into single-rate DAGs. Beyond the focus on upper-bounding MRT and MDA [8, 17], Jiang et al. [13] examined maximum time disparity (MTD)—typically used to measure deviations in sensor sampling times—between multi-sensor data, emphasizing the need for accurate synchronization for fusion. For enhanced cause-effect chains, Tang et al. [28] reduced MRT via buffer limits and data refreshing, and later proposed a dynamic priority inheritance and buffer manipulation protocol to lower MRT and MDA in their sporadic variant of these chains [30]. Regarding upper-bounding MTD alongside end-to-end latency, Sun et al. [26] proposed a mechanism to select which sensor data is used by an actuator and fused along the path from sensor to actuator, assuming fusion nodes are triggered only when a new data arrives.

Data communication and fusion in cause-effect chains are also addressed in ROS 2-related studies. Li et. al. [16] explored the potential and limitations of ROS 2 message synchronizer in multi-sensor data fusion, while Sun et. al. [27] proposed a novel message synchronization policy for ROS 2 to improve MTD when fusing multi-sensor data. Saito et. al. [21] proposed a priority-based message transmission and a synchronization node to address MTD of sensor data. Later, in [22], they introduced a synchronization system that buffers the highest-rate sensor to align periods, converting multi-rate DAGs into single-rate ones. Sun et al. [25] proposed an ILP-based model to optimize MRT and MDA by reducing redundant workload and unnecessary messages. For enhanced cause-effect chains, the model in [9–11, 31, 32] stands as one of the most advanced implementations. They analyzed MRT and MDA within a single-threaded executor setup by classifying intra- and inter-task data communication patterns [32] and later demonstrated cases in which MRT and MDA can be equivalent [11]. This analysis was extended to multiple single-threaded executors, incorporating additional

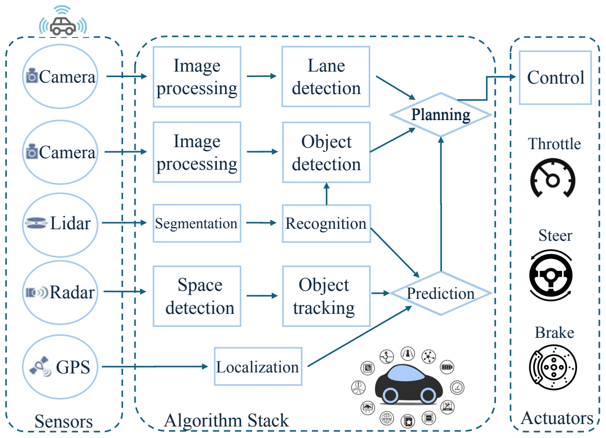


Fig. 1: Holistic overview of the ADS software stack, inspired by [15, 36].

communication paradigms [31]. Yano et al. [38] proposed a scheduling model that decomposes the Autoware DAG into multi-deadline sub-DAGs, aiming to bypass challenges related to synchronization, queue consumption patterns, and intricate data dependencies.

In summary, existing approaches do not fully capture the complexity of real-world ADS that feature multiple triggering options for fusion nodes and branch-then-fusion paths. While each prior work addresses specific fusion patterns, none provides a comprehensive framework that handles all fusion behaviors found in practice (more details in Sec. 4). Moreover, integrating these approaches to analyze complete ADS is non-trivial, since each work employs different assumptions, abstractions, and analytical approaches, making them often incompatible with one another. This highlights a gap in modeling fusion nodes within complex ADS structures, which motivates our work.

3 Background and System Model

3.1 Overview of ADS software stack

ADS consists of tightly integrated software components, each essential to the vehicle's safe operation. As shown in Fig. 1, these components can be categorized into three main classes: sensors, algorithm stack, and control systems (actuators). *Sensors*, including LiDAR, radar, cameras, GPS, and others, gather real-time data from the environment, which serves as input for the system's perception. The *algorithm stack*, which includes object and lane detection, localization, prediction, planning, etc., processes sensor data and interprets it to real-time situational-aware decisions. Finally, *actuators* execute these decisions to control vehicle dynamics, including steering, acceleration, and braking.

Communication between these interdependent software components happens through a deep processing pipeline. Data sampled by sensors is processed and, at various points along the pipeline, combined with data from other sensors or outputs from intermediate components in the algorithm stack, such as prediction or planning nodes, as shown in Fig. 1. These nodes, called fusion nodes, integrate multiple inputs to produce outputs, with their activation timing and triggering patterns varying based on the ADS application's specifications. The interaction between components, the diverse data fusion patterns, and the presence of branch-then-fusion paths create a complex structure that demands a robust framework to manage real-time constraints, as delays in any component can compromise vehicle safety. Motivated by this, the real-time systems community has introduced several key metrics to ensure timely responses.

- **Maximum Reaction Time (MRT):** The longest time interval from the occurrence of an external event (which can happen between sensor instances) to the first actuator response to that event [8].[1] For a DAG with multiple sensors contributing to an actuator, MRT should consider the maximum value across all sensor-to-actuator paths.
- **Maximum Time Disparity (MTD):** The maximum time difference between the release times of the oldest and newest sensor data contributing to an actuator output instance, measured across all instances of that actuator [13].
- **Worst-Case Response Time (WCRT):** The maximum duration of each chain in a DAG, measured from a sensor task's release time to an actuator task's completion time [23].
- **Peak Age of Information (PAoI):** AoI represents the time elapsed since the generation of the latest data sample to the current time [35]. PAoI is the maximum AoI value across all sensors in the system [5].

3.2 System Model

This section presents our system model, defining DAG, chain, and task representations on a multi-core platform with identical CPU cores. The summary of notations is provided in Table 1.

DAG. A Directed Acyclic Graph (DAG) is used to represent the ADS software application. We denote the DAG as $G = (V, E)$, where V represents the set of vertices and E represents the set of edges. Each vertex or node in the DAG corresponds to a task, and the terms "vertex", "node", and "task" are used interchangeably throughout the paper. Each edge $E_{i',i} \in E$ indicates a data dependency, where the output of task i' serves as the input to task i. Without loss of generality, we assume inter-task communication occurs through a unit-sized buffer between consecutive tasks, following the "last-is-best" principle [17], which ensures that only the most recent data is retained. Additionally, the buffer is non-blocking [1], which is easily achievable through double-buffering techniques within the implicit communication paradigm [17]. Source and sink nodes of the DAG, respectively, are the representative of tasks from the sensor and actuator classes mentioned in Sec. 3.1. We assume the DAG under analysis, G, comprises m sensor nodes and n non-sensor nodes, with a total size of $|V| = m + n$.

[1] A recent study has reported that MRT is equivalent to the maximum data age (MDA) [11]; hence, we use MRT as a representative of both in this work.

Table 1: Table of Notations

Symbol	Description
$G = (V, E)$	DAG G with the set of vertices V and edges E
τ_i	The task i where $\tau_i \in V$
e_i, T_i, D_i	The execution time, the period, and the deadline of τ_i
θ_i	The type of τ_i where $\theta_i \in$ {"sen", "sub", "t-fus", "w-fus", "i-fus"}.
$pred_i$	The set of adjacent predecessors of τ_i where $pred_i = \{\tau_{i'} \vert \forall E_{i',i} \in E\}$
$\tau_{i,j}$	The j-th instance of τ_i
$s_{i,j}, f_{i,j}, d_{i,j}$	The start time, finish time, and absolute deadline of instance $\tau_{i,j}$
$m, n, \mathit{\Pi}$	The number of sensors, non-sensor tasks, and cores
$\tau_s, \tau_\otimes$	A source and a sink node in the DAG

Chain. A chain represents a path within the DAG, running from a source node (sensor) to a sink node (actuator) and capturing a sequence of dependent tasks in a set order. Together, these chains form the complete DAG, showing all interdependencies in the system. Since our focus is on the overall DAG structure rather than individual chains, we use DAG notations.

Tasks. A task τ_i is a software component represented as a node in the DAG (i.e., $V = \{\tau_1, \ldots, \tau_{m+n}\}$), which can be triggered either by a *timer* or by an external *event*. The task τ_i is characterized as $\tau_i = \{e_i, T_i, D_i, \theta_i, pred_i\}$; where e_i is the worst-case execution time (WCET) of τ_i; T_i and D_i denote its period and relative deadline, respectively; θ_i represents the type of τ_i (e.g., sensor, fusion, etc; details will be followed); and $pred_i$ is the set of adjacent predecessor tasks of τ_i in the DAG, i.e., $pred_i = \{\tau_{i'} \vert \forall E_{i',i} \in E\}$. Note that for an event-triggered task, we define $T_i = 0$. We assume that the execution of each task follows the *read-execute-write* semantics, where input data is read at the task's start time, and output data is written at its finish time [3, 6, 32]. Tasks are scheduled non-preemptively, like *callbacks* in ROS 2 and *runnables* in AUTOSAR.

Task Instances. Each task, whether triggered by a timer or an event, generates a sequence of task instances. We denote the j-th instance of task τ_i as $\tau_{i,j}$. The start and finish times of execution of $\tau_{i,j}$ are represented by $s_{i,j}$ and $f_{i,j}$, respectively. For simplicity, we assume that a task instance is released (ready for execution) immediately upon being triggered, and its absolute deadline, denoted as $d_{i,j}$, is calculated by this release time $r_{i,j}$ plus its relative deadline D_i. Also, the first instance of each sensor task is assumed to arrive at time 0.

4 Task Types and Fusion Patterns in ADS

In this section, we characterize the diverse behaviors of various types of tasks in ADS, with a particular focus on fusion tasks. Based on various triggering factors and the number of precedent inputs, we categorize a task τ_i into one of five distinct types, which are detailed in the following paragraph.

– Sensor (θ_i = "sen"): Sensor task is a periodic timer-triggered task, sampling real-time data and generating output for a successor task. Considering a sensor task is a source node in DAG, it has no predecessor dependencies. Therefore, $pred_i = \varnothing$ is an empty set for a sensor task τ_i.

- Subscription ($\theta_i =$"sub"): Subscription task is event-triggered, activated upon receiving one input message and publishing one output message accordingly. For these tasks, the size of the $pred_i$ is always 1, meaning $\|pred_i\| = 1$.
- T-fusion ($\theta_i =$ "t-fus"): T-fusion task is **t**imer-triggered and periodic, gathering inputs from multiple preceding tasks and merging them into a unified output. A real-world ADS example of this type can be found in Autoware, such as the `ndt_scan_matcher` component [2]. Notably, an intermediate or sink node in a DAG that is timer-triggered but has only one input can also be considered a special form of T-fusion.
- W-fusion ($\theta_i =$ "w-fus"): W-fusion is a **w**ait-for-all fusion task, triggered when it has received inputs from all its predecessor tasks, only generating an output once all required inputs are available. A real-world ADS example of this type in Autoware is `tracking_object_merger` component [2].
- I-fusion ($\theta_i =$ "i-fus"): In contrast to W-fusion, an I-fusion task is triggered as **i**mmediately as any of its predecessor tasks publishes a new message, reacting without waiting for all inputs to arrive. An example of this type in Autoware is the `traffic_light_occlusion_predictor` component [2].

Among these task types, sensor and subscription nodes have been extensively studied in prior work. Their modeling and scheduling are relatively straightforward, and we include them here for the sake of completeness. However, unlike prior work where each study considered only one type of fusion node, our main focus is on all three fusion types described above. For example, Toba et al. [33] considered *trigger* and *update* edges, where only *trigger* edges activate the fusion node's execution. Similarly, Teper et al. [32] used *passive* and *trigger* edges, with the fusion node triggered solely by the one dominant trigger edge. These behaviors fall into our W-fusion category, which can also handle cases where the triggering edge is not predetermined. Multi-rate DAG analysis [17, 25] assumes that all fusion nodes are time-triggered, which can be represented using our T-fusion nodes. Sun et al. [26] assumed that only sensors and actuators are timer-triggered and all other intermediate nodes, including fusion nodes, are event-triggered, which our I-fusion node can model.

To better illustrate these task types, Fig. 2 presents an example DAG with the node types defined above. Nodes τ_1 to τ_4 represent sensors, τ_5 and τ_9 are subscription nodes, and τ_8, τ_{10}, and τ_{11} correspond to I-fusion, T-fusion, and W-fusion nodes, respectively. Our model supports not only multiple sink nodes but also branch (or fork) nodes. A branch node whose output diverges into multiple paths can be any of the above node types. If the immediate successor of a branch node is a subscription node, e.g., τ_6 and τ_7, it is modeled as "i-fus" for formulation purposes (for details, see Appendix A.4 of our extended manuscript [24]).

Based on task types, the number of instances of each task can be calculated. Consider the time interval of interest Δ that is an integer multiple of the hyperperiod (HP), i.e., $\Delta = k \cdot HP$ where HP is the least common multiple (LCM) of all time-triggered tasks (sensors and T-fusion nodes). The interval Δ

starts at time 0, and all sensor tasks initially arrive at time 0. Alg. 1 details the calculation of the instance count, *n-ins*, for a task τ_i within this interval $\varDelta$. For timer-triggered tasks like sensor and T-fusion nodes, the instance count is straightforward to compute by $n\text{-}ins(\tau_i, \varDelta) = \frac{\varDelta}{T_i}$ because $\varDelta$ is an integer multiple of HP (lines 2-3). For event-triggered tasks, the instance count calculation varies. The instance count of a W-fusion node is determined by the least frequent adjacent predecessor's *n-ins* (lines 4-5), as the W-fusion has to wait for all predecessors by its definition. An I-fusion node is triggered by any updated input from its predecessors. But at the very beginning, it still has to wait for every predecessor to produce at least one input for fusion. Hence, the instance count of an I-fusion node (lines 6-7) is obtained by the sum of instance counts of its predecessors ($\sum n\text{-}ins(pre, \varDelta)$) subtracted by the number of predecessor instances to wait ($\|pred_i\| - 1$). In the case of a subscription task, the instance count matches the *n-ins* of its adjacent predecessor, as it has a single predecessor and is triggered whenever input is received from its predecessor. (lines 8-9).

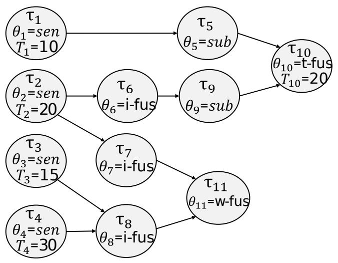


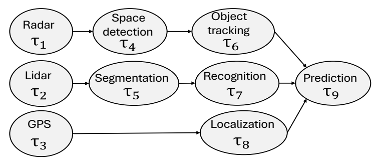
Fig. 2: Task types in a DAG

Algorithm 1 n-ins(τ_i,$\varDelta$)

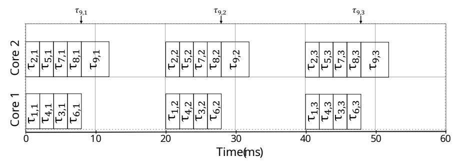
```
1: $n\text{-}ins \leftarrow 0$
2: **if** $\theta_i \in \{\text{“sen”}, \text{“t-fus”}\}$ **then**
3:     $n\text{-}ins \leftarrow \frac{\varDelta}{T_i}$
4: **else if** $\theta_i ==$ “w-fus” **then**
5:     $n\text{-}ins \leftarrow \min_{pre \in pred_i} \big(n\text{-}ins(pre, \varDelta)\big)$
6: **else if** $\theta_i ==$ “i-fus” **then**
7:     $n\text{-}ins \leftarrow$
           $\big(\sum_{pre \in pred_i} n\text{-}ins(pre, \varDelta)\big) - (\|pred_i\| - 1)$
8: **else if** $\theta_i ==$ “sub” **then**
9:     $n\text{-}ins \leftarrow n\text{-}ins(pred_i, \varDelta)$
10: **end if**   **return** $n\text{-}ins$
```

Example: Instance Counts. Let us apply Alg. 1 to the DAG example in Fig. 2, assuming $\varDelta = 1 \cdot HP = 60$. For the sensor tasks $\tau_1, \tau_2, \tau_3, \tau_4$, Alg. 1 gives $n\text{-}ins(\tau_1, HP) = 6$, $n\text{-}ins(\tau_2, HP) = 3$, $n\text{-}ins(\tau_3, HP) = 4$, $n\text{-}ins(\tau_4, HP) = 2$. For the subscription task τ_5, it inherits τ_1, so $n\text{-}ins(\tau_5, HP) = n\text{-}ins(\tau_1, HP) = 6$. For the I-fusion tasks with single predecessors, τ_6 and τ_7, Alg. 1 gives $n\text{-}ins(\tau_6, HP) = n\text{-}ins(\tau_2, HP) - (1 - 1) = 3$ and $n\text{-}ins(\tau_7, HP) = n\text{-}ins(\tau_2, HP) - (1 - 1) = 3$, showing that they are triggered by every instance of their predecessors. For the other I-fusion task τ_8, $n\text{-}ins(\tau_8, HP) = n\text{-}ins(\tau_3, HP) + n\text{-}ins(\tau_4, HP) - (2 - 1) = 5$ because the first instance $\tau_{8,1}$ is triggered only after both $\tau_{3,1}$ and $\tau_{4,1}$ have arrived. For the subscription task τ_9, $n\text{-}ins(\tau_9, HP) = n\text{-}ins(\tau_6, HP) = 3$. For the T-fusion task τ_{10}, $n\text{-}ins(\tau_{10}, HP) = 3$. For the W-fusion task τ_{11}, $n\text{-}ins(\tau_{11}, HP) = min(n\text{-}ins(\tau_7, HP), n\text{-}ins(\tau_8, HP)) = min(3, 5) = 3$ because τ_{11} waits for new inputs from both τ_7 and τ_8 before triggering a new instance.

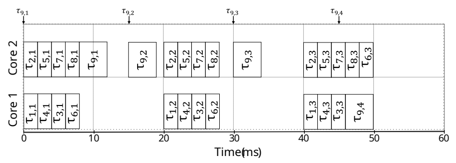
Example: Scheduling Behavior. To clarify the behavior of fusion nodes in the scheduling context, we provide illustrative examples for the three fusion types, shown in Fig. 3a, which is a subset of the ADS software stack in Fig. 1. In this DAG, the *prediction* node serves as a fusion node, while all other non-sensor



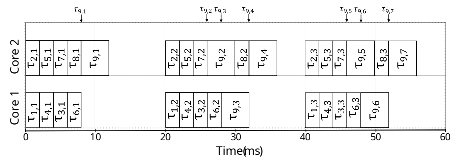
(a) DAG structure



(c) DAG scheduling with W-fusion node



(b) DAG scheduling with T-fusion node



(d) DAG scheduling with I-fusion node

Fig. 3: Illustrative examples on fusion nodes

tasks are subscription nodes. We set the WCET of sensor and subscription nodes to 2 ms and assigned a period of 20 ms to each sensor. The WCET of the fusion node was set to 4 ms, and we explored its behavior by varying its type among T-fusion, W-fusion, and I-fusion. We first configure the *prediction* task as a T-fusion node with a period of 15 ms and examine its effect on DAG scheduling, as shown in Fig. 3b. In this case, the *prediction* task is triggered once per its period (= 15), regardless of input arrivals. Next, to illustrate W-fusion, we change the *prediction* task to have the type θ = "w-fus". The scheduling behavior of W-fusion, waiting for all its dependencies before execution, is depicted in Fig. 3c. Finally, we demonstrate I-fusion by setting the *prediction* task to have θ = "i-fus". As shown in Fig. 3d, I-fusion triggers the *prediction* task immediately upon the arrival of any input, except the first instance, highlighting its responsiveness compared to the other fusion types. The first instance of each fusion node requires all its predecessors to be executed at least once. The arrows in the figures indicate the triggering time of each fusion instance.

Now that we have a clearer understanding of fusion tasks and their impact on scheduling and real-time performance, we formulate their diverse behaviors and analyze their effects on key metrics such as MRT, MTD, PAOI, and WCRT in the following section.

5 Formulation of Fusion Tasks

In this section, we begin by providing a brief formulation of the general requirements for resource allocation and task scheduling in a multi-core system, using ILP constraints. We then focus on task-specific behaviors, particularly fusion tasks, and explore how these tasks influence scheduling and timing, especially their impact on the start and finish times of other tasks. Finally, we explain how to model and formulate key real-time performance metrics for ADS into ILP, enabling the study of the effects of different fusion types on these metrics.

The DAG under analysis, G, operates on a multi-core platform equipped with Π identical CPU cores. We do not restrict tasks to be bound to specific cores, allowing instances of the same task to execute on different cores. To achieve this, we define a binary variable $y^{\pi}_{i,j}$ that equals 1 if an instance $\tau_{i,j}$ is assigned to core π (where $1 \leq \pi \leq \Pi$), and 0 otherwise. In a scheduling problem, once a task instance is assigned to a core, its start and finish times need to be determined. Since tasks are non-preemptible, each task runs uninterrupted once it starts. Therefore, given the start time, we can calculate the finish time by adding the WCET e_i of the task. As a result, the scheduling problem reduces to determining the start time of each task instance; once the start time is known, the finish time follows directly. To ensure each task instance $\tau_{i,j}$ meets its deadline, we compare its finish time $f_{i,j}$ with its absolute deadline $d_{i,j}$. Since expressing basic scheduling requirements, such as task-to-core mapping, overlap prevention, task deadline constraints (D_i and $d_{i,j}$), in ILP is well-known and not the main focus of our work, we discuss it in Appendix A of [24].

Before focusing on how to formulate the different fusion types into ILP constraints, we briefly review sensor and subscription nodes for the sake of completeness. For sensor nodes, which have no preceding data dependencies, the start time is determined solely by their timer period and instance release time, following the standard definition of periodic tasks. To ensure that a sensor instance $\tau_{i,j}$ (where $\theta_i =$ "sen") starts after its release time and finishes before the next instance arrives, we define the following constraints:

$$\begin{aligned} s_{i,j} &\geq T_i \cdot (j-1), && \forall \tau_i \in V \wedge\ \theta_i = \text{``sen''}, \forall j \in [1, \textit{n-ins}(\tau_i, \Delta)] \\ s_{i,j} &\geq f_{i,j-1}, && \forall \tau_i \in V \wedge\ \theta_i = \text{``sen''}, \forall j \in [1, \textit{n-ins}(\tau_i, \Delta)] \end{aligned} \tag{1}$$

For subscription nodes, the start time is determined by their dependency on a single predecessor node. Specifically, the start time of a subscription task instance must follow the finish time of its preceding task instance with the same instance index. Therefore, for a subscription task instance $\tau_{i,j}$ (where $\theta_i =$ "sub"), the following equation holds, with $\tau_{i'}$ representing its sole predecessor:

$$s_{i,j} \geq f_{i',j}, \quad \forall \tau_i \in V \wedge\ \theta_i = \text{``sub''},\ \textit{pred}_i = [\tau_{i'}], \forall j \in [1, \textit{n-ins}(\tau_i, \Delta)] \tag{2}$$

5.1 Fusion Task Constraints

Unlike subscription tasks, fusion tasks (like T-fusion, W-fusion, or I-fusion) have multiple predecessors with different arrival rates. Hence, it is not straightforward to establish the relationship between the instance index of a fusion task and those of its predecessors.

To address this issue, we first introduce the concept of *producer* tasks. Producer tasks are sensor and fusion nodes; they generate and provide data to subsequent tasks, establishing the root for a new instance for subsequent non-producer tasks. Hence, any non-producer tasks (i.e., subscription tasks) use the same instance indices as those of their producers' instances.

Let us use $\textit{producer}(\tau_i)$ to denote the producer task of τ_i. If τ_i is a subscription task, $\textit{producer}(\tau_i)$ obviously gives only one task: the closest preceding sensor or fusion task of τ_i in the DAG hierarchy. If τ_i is either a sensor or fusion task, $producer(\tau_i) = \tau_i$ because τ_i is a producer by itself. Note that $producer(\tau_i)$ should not be confused with $pred_i$: if τ_i is a fusion task, it can have multiple predecessors ($|pred_i| \geq 1$) and each of its predecessors has its own producer ($producer(\tau_{i'})|\tau_{i'} \in pred_i$). We also define $pred\_producers(\tau_i)$ as the set of producers of τ_i's predecessors:

$$pred\_producers(\tau_i) = \{producer(\tau_{i'})|\forall \tau_{i'} \in pred_i\}$$

For example, consider the DAG in Fig. 3a. The sink node τ_9 has $pred_9 = \{\tau_6, \tau_7, \tau_8\}$ and $pred\_producers(\tau_9) = \{\tau_1, \tau_2, \tau_3\}$. Note that $producer(\tau_9) = \tau_9$ because τ_9 is a fusion node.

Next, to represent which instances of predecessors contribute to a fusion task instance, we define a binary decision variable as follows:

Definition 1. *Consider a fusion task τ_i and one of its predecessors' producer τ_p where $\tau_p \in pred\_producers(\tau_i)$. A binary decision variable $u_{(i,j),(p,j_p)}$ indicates whether the fusion task's instance $\tau_{i,j}$ uses the j_p-th instance of τ_p:*

$$u_{(i,j),(p,j_p)} = \begin{cases} 1, & \textit{if } \tau_{i,j} \textit{ uses the } j_p\textit{-th instance of } \tau_p \\ 0, & \textit{otherwise} \end{cases} \tag{3}$$

With this definition, for a fusion task instance $\tau_{i,j}$, we can determine which instance j' of $producer(\tau_{i'} \in pred_i)$ is used by $\tau_{i,j}$. Recall that $producer(\tau_{i'})$ determines the instance index of the subsequent task $\tau_{i'}$; this applies to any type of task, because if τ_i' is either a sensor or fusion task, $producer(\tau_{i'}) = \tau_i'$. Therefore, we can set constraints on the start time of the fusion task instance $\tau_{i,j}$ by linking it to the finish time of the predecessor instance $\tau_{i',j'}$ it uses. This ensures that each fusion instance begins only after the appropriate predecessor instance completes.

Hence, we establish the following lemma for all three types of fusion nodes.

Lemma 1 (Constraint: start time of fusion nodes). *The start time of a fusion task instance $\tau_{i,j}$, i.e., $\theta_i \in \{$"t-fus", "w-fus", "i-fus"$\}$, should satisfy the following condition:*

$$s_{i,j} \geq f_{i',j'} \cdot u_{(i,j),(producer(\tau_{i'}),j')}, \ \forall \tau_{i'} \in pred_i, \forall j' \in [1, n\text{-}ins(\tau_{i'}, \varDelta)] \tag{4}$$

Proof. Here, $\tau_{i'}$ is one of τ_i's predecessors ($\forall \tau_{i'} \in pred_i$), and $u_{(i,j),(producer(\tau_{i'}),j')}$ indicates whether $\tau_{i,j}$ uses j'-th instance of $\tau_{i'}$'s producer. To prove this equation, we consider two cases for $u_{(i,j),(producer(\tau_{i'}),j')}$: (i) If $u_{(i,j),(producer(\tau_{i'}),j')} = 0$, the right-hand side is zero, which is always valid as start times are never negative. (ii) If $u_{(i,j),(producer(\tau_{i'}),j')} = 1$, meaning the j'-th instance of $\tau_{i'}$'s producer is used by $\tau_{i,j}$, we have two sub-cases:
(ii-A) If $\tau_{i'}$ is a sensor or fusion node, $producer(\tau_{i'}) = \tau_{i'}$, and $u_{(i,j),(producer(\tau_{i'}),j')} = u_{(i,j),(i',j')} = 1$. This means $\tau_{i',j'}$ contributes to $\tau_{i,j}$, so $s_{i,j} \geq f_{i',j'}$.
(ii-B) If $\tau_{i'}$ is a subscription node, it inherits the instance index of its producer. Thus, the j'-th instance of $\tau_{i'}$ is used by $\tau_{i,j}$, and $s_{i,j} \geq f_{i',j'}$ holds. □

Each instance of a fusion task uses only one instance from each of its predecessors' producers, specifically the most recent instance available from each producer. This imposes the following constraints.

Lemma 2 (Constraint: used only once by an instance). *Since a fusion task instance $\tau_{i,j}$ uses only one instance from each of its predecessors' producers, the following constraint must hold for any $\tau_i \in V$, where $\theta_i \in \{$"t-fus", "w-fus", "i-fus"$\}$, and any $j \in [1, n\text{-}ins(\tau_i, \Delta)]$:*

$$\sum_{j'=1}^{n\text{-}ins(\tau_{i'},\Delta)} u_{(i,j),(producer(\tau_{i'}),j')} = 1, \ \forall \tau_{i'} \in pred_i \tag{5}$$

Proof. We prove by contradiction. Assume for a fusion instance $\tau_{i,j}$ and its predecessor $\tau_{i'}$, the summation can be other than 1. We consider two cases:
(i) $\sum_{j'=1}^{n\text{-}ins(\tau_{i'},\Delta)} u_{(i,j),(producer(\tau_{i'}),j')} = 0$: This means $u_{(i,j),(producer(\tau_{i'}),j')} = 0$ for all j', meaning no instance of $producer(\tau_{i'})$ or $\tau_{i'}$ contributes to $\tau_{i,j}$. This contradicts our system model, which requires that at least one of the predecessors must execute to perform a fusion operation.
(ii) $\sum_{j'=1}^{n\text{-}ins(\tau_{i'},\Delta)} u_{(i,j),(producer(\tau_{i'}),j')} > 1$: If the summation is greater than 1, multiple instances of $\tau_{i'}$ contribute to a single fusion instance $\tau_{i,j}$. This contradicts fusion node requirements, as each fusion node uses only the most recent instance from each predecessor. While some instances may be dropped (T-fusion or W-fusion) or reused (T-fusion or I-fusion), only one instance of $\tau_{i'}$ should contribute to $\tau_{i,j}$.

Thus, by contradiction, the summation must be 1, proving the lemma. □

Lemma 3 (Constraint: most recent instance is used). *Since a fusion task instance $\tau_{i,j}$ uses the most recent instance from each of its predecessor tasks' producers, the following constraint must hold for any $\tau_i \in V$, where $\theta_i \in \{$"t-fus", "w-fus", "i-fus"$\}$, and any $j \in [1, n\text{-}ins(\tau_i, \Delta)]$:*

$$\sum_{j'=1}^{n\text{-}ins(\tau_{i'},\Delta)} j' \cdot u_{(i,j),(producer(\tau_{i'}),j')} \leq \sum_{j'=1}^{n\text{-}ins(\tau_{i'},\Delta)} j' \cdot u_{(i,j+1),(producer(\tau_{i'}),j')}, \ \forall \tau_{i'} \in pred_i \tag{6}$$

Proof. By Lemma 2, for any fusion instance $\tau_{i,j}$ and predecessor $\tau_{i'}$, only one instance j' in $[1, n\text{-}ins(\tau_{i'}, \Delta)]$ has $u_{(i,j),(producer(\tau_{i'}),j')} = 1$, with all other terms being 0. Multiplying each term by j', we get $j' \cdot u_{(i,j),(producer(\tau_{i'}),j')} = j'$ for the contributing instance and 0 for others. Summing over all instances gives the instance index j' of the single contributing instance, which represents the instance of $\tau_{i'}$ used in $\tau_{i,j}$. The inequality ensures that the instance of $\tau_{i'}$ used in $\tau_{i,j+1}$ is the same or newer than in $\tau_{i,j}$. □

Having established the constraints for all fusion node types, we now outline the specific constraints for each type.

T-fusion: For T-fusion tasks, which are timer-triggered, Equation (1) applies, along with the fusion-specific lemmas mentioned above.

W-fusion: For W-fusion tasks, where an instance is triggered only after receiving inputs from all its predecessor tasks, an instance of each predecessor's producer can be used at most once by the fusion task instances. This leads to the following constraint.

Lemma 4 (Constraint: used once by "w-fus"). *Since, for a fusion task τ_i, an instance of each predecessor's producer can be used at most once by the fusion task instances, the following constraint must hold for any $\tau_i \in V$ with $\theta_i =$ "w-fus":*

$$\sum_{j=1}^{n\text{-}ins(\tau_i,\Delta)} u_{(i,j),(producer(\tau_{i'}),j')} \leq 1, \quad \forall \tau_{i'} \in pred_i, \forall j' \in [1, n\text{-}ins(\tau_{i'},\Delta)] \tag{7}$$

Proof. We prove by contradiction. Assume for a W-fusion task τ_i and its predecessor $\tau_{i',j'}$, the summation value is greater than 1. This implies at least two instances of τ_i use the same j'-th instance of $\tau_{i'}$, which contradicts the W-fusion property. A W-fusion waits for all predecessor inputs to arrive, and once $\tau_{i',j'}$ is used, the next instance of τ_i must wait for a new instance of $\tau_{i'}$, not reuse $\tau_{i',j'}$.

Unlike W-fusion, T-fusion and I-fusion tasks can use the same producer instance multiple times across different instances of the fusion task.

I-fusion: In the case of I-fusion tasks, an instance of the fusion node is triggered by the arrival of *any* instance of its predecessor tasks. The key distinction here is that, for I-fusion, at least one of the predecessor tasks must provide a new instance to trigger an I-fusion task instance.

Lemma 5 (Constraint: at least one new instance for "i-fus"). *The following constraint ensures that the I-fusion task τ_i progresses from instance $\tau_{i,j}$ to $\tau_{i,j+1}$, only when at least one of its predecessors' producer has triggered a new instance; where $\tau_i \in V$ with $\theta_i =$ "i-fus" and $j \in [1, n\text{-}ins(\tau_i,\Delta)]$:*

$$\sum_{\tau_{i'} \in pred_i} \Big(\sum_{j'=1}^{n\text{-}ins(\tau_{i'},\Delta)} j' \cdot u_{(i,j+1),(producer(\tau_{i'}),j')} - \sum_{j'=1}^{n\text{-}ins(\tau_{i'},\Delta)} j' \cdot u_{(i,j),(producer(\tau_{i'}),j')} \Big) \geq 1 \tag{8}$$

Proof. For a fusion instance $\tau_{i,j}$ and its predecessor $\tau_{i'}$, the term $\sum_{j'=1}^{n\text{-}ins(\tau_{i'},\Delta)} j' \cdot u_{(i,j),(producer(\tau_{i'}),j')}$ gives the instance index of $\tau_{i'}$ used in $\tau_{i,j}$. Similarly, for $\tau_{i,j+1}$, the expression $\sum_{j'=1}^{n\text{-}ins(\tau_{i'},\Delta)} j' \cdot u_{(i,j+1),(producer(\tau_{i'}),j')}$ gives the instance index of $\tau_{i'}$ used in that fusion instance. According to Lemma 3, the difference between these two sums must be greater than or equal to 0. If the difference is 0, the same instance of $\tau_{i'}$ is used in both $\tau_{i,j}$ and $\tau_{i,j+1}$. If the difference is positive, a newer instance is used in $\tau_{i,j+1}$. To ensure that at least one predecessor triggers the I-fusion node with a newer instance, we sum these differences across all predecessors and confirm that the total is non-zero. □

5.2 Real-Time Performance Metric Constraints

Based on the formulation of fusion task behavior in Sec. 5.1, this section explains how to model and derive key real-time performance metrics—such as MRT, MTD, PAoI, and MS—using our proposed ILP optimization method. These metrics are essential for assessing system responsiveness and ensuring optimized performance. As they reflect the timing relationships between source (sensor, τ_s) and sink (actuator, $\tau_\otimes$) nodes in the DAG, we must identify which instances of sensor nodes are linked to each instance j of a sink node $\tau_\otimes$, i.e., $\tau_{\otimes,j}$, when there is a directed path from a sensor τ_s to $\tau_\otimes$.

To determine which instances of sensor nodes contribute to each fusion or sink node instances, we define an additional binary variable $U_{(i,j),(s,j_s)}$, where $\tau_{i,j}$ is the j-th instance of task τ_i (either a sink or fusion node) and τ_{s,j_s} is the j_s-th instance of a sensor task τ_s. Specifically:

$$U_{(i,j),(s,j_s)} = \begin{cases} 1, & \text{if instance } \tau_{i,j} \text{ uses the } j_s\text{-th instance of sensor } \tau_s \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

Note that U differs from the decision variable u (Def. 1). While $u_{(i,j),(p,j_p)}$ represents direct relationships between fusion instances and their immediate predecessors' producers (determined by the ILP solver), $U_{(i,j),(s,j_s)}$ captures the end-to-end relationships between any task instance and sensor instances (computed using the u variables). The computation of $U_{(i,j),(s,j_s)}$ is done as follows.

1. If τ_i is a sink and subscription node. It means τ_i has only one immediate preceding producer, i.e., $producer(\tau_i) = \tau_p$.
   - If τ_i is directly connected to a sensor τ_s ($\tau_p = \tau_s$), every j-th instance of τ_i is triggered by j-th instance of τ_s. Hence, $U_{(i,j),(s,j)} = 1$
   - If τ_i's producer, τ_p, is a fusion node, we have:

     $$U_{(i,j),(s,j_s)} = U_{(p,j),(s,j_s)}, \quad \forall \tau_s \in V, \theta_s = \text{“}sen\text{”}, j_s \in [1, n\text{-}ins(\tau_s, \varDelta)]$$

     This is obvious because any sensor instance τ_{s,j_s} used by $\tau_{p,j}$ is in turn used by $\tau_{i,j}$.
2. If τ_i is a fusion node (either sink or intermediate node). Since τ_i may have multiple preceding producers through τ_i's predecessors, we need to consider the following cases for all $\tau_p \in pred\_producers(\tau_i)$.
   - If τ_p is a sensor τ_s ($\tau_p = \tau_s$), we can directly use $u_{(i,j),(s,j_s)}$:

     $$U_{(i,j),(s,j_s)} = u_{(i,j),(s,j_s)}, \quad \forall j_s \in [1, n\text{-}ins(\tau_s, \varDelta)]$$

   - If τ_p is a fusion node, we should take into account whether any instance of τ_p uses τ_{s,j_s} or not. Also, we need to consider all paths that may exist between τ_i and τ_s, as the DAG may contain branch nodes, resulting in multiple paths. These can be determined by the following lemma.

Lemma 6 (Constraint: U for fusion nodes). *The binary variable $U_{(i,j),(s,j_s)}$ for any fusion node τ_i and any sensor node τ_s is equal to 1 if the following inequality holds:*

$$\sum_{\tau_p \in pred\_producers(\tau_i)} \sum_{j_p=1}^{n\text{-}ins(\tau_p,\Delta)} u_{(i,j),(p,j_p)} \cdot U_{(p,j_p),(s,j_s)} \geq 1 \tag{10}$$

where $j \in [1, n\text{-}ins(\tau_i, \Delta)]$, and $j_s \in [1, n\text{-}ins(\tau_s, \Delta)]$.

Proof. *Consider an instance $\tau_{i,j}$ that uses data from sensor instance τ_{s,j_s}. This data flows through an intermediate fusion node τ_p. For $\tau_{i,j}$ to use data from τ_{s,j_s}, there must exist some instance j_p of τ_p such that: (i) $\tau_{i,j}$ uses τ_{p,j_p} (represented by $u_{(i,j),(p,j_p)} = 1$), and (ii) τ_{p,j_p} uses τ_{s,j_s} ($U_{(p,j_p),(s,j_s)} = 1$). The product $u_{(i,j),(p,j_p)} \cdot U_{(p,j_p),(s,j_s)}$ equals 1 if and only if both conditions are met for instance j_p. The summation over all possible instances j_p of τ_p captures every potential instance through which data from τ_{s,j_s} could reach $\tau_{i,j}$. Similarly, the summation over all $\tau_p \in pred\_producers(\tau_i)$ ensures that all possible paths in the DAG from τ_{s,j_s} to $\tau_{i,j}$ are considered. Therefore, $U_{(i,j),(s,j_s)} = 1$ if there exists at least one instance j_p of τ_p on at least one path from τ_{s,j_s} to $\tau_{i,j}$, which is exactly what the ≥ 1 condition represents.* □

With the calculation of U, we can determine, for each sensor that has at least a path to a sink, exactly which instances of the sensor τ_s contribute to each instance of the sink node $\tau_\otimes$. However, multiple sensors may converge through fusion nodes to contribute data to a sink node. Among the end-to-end latency metrics discussed, MRT needs to identify the oldest data among all contributing sensor instances for a given sink instance $\tau_{\otimes,j}$, i.e., the earliest released sensor instance. To find this earliest release time among sensor instances contributing to a task instance $\tau_{i,j}$, we define the function $OTS(\tau_{i,j})$ as follows:

$$\begin{aligned} OTS(\tau_{i,j}) = min\big\{&T_s \cdot (j_s - 1) | \forall \tau_s \in V, \theta_s = \text{“sen”}, \\ &\forall j_s \in [1, n\text{-}ins(\tau_s, \Delta)], \quad \text{if } U_{(i,j),(s,j_s)} = 1\big\} \end{aligned} \tag{11}$$

Similarly, to compute the MTD metric for a sink node, in addition to $OTS(\tau_{i,j})$, we must identify the newest data among all contributing sensor instances for the given sink instance $\tau_{\otimes,j}$. To capture this latest release time among sensor instances contributing to $\tau_{i,j}$, we define the function $NTS(\tau_{i,j})$ as follows:

$$\begin{aligned} NTS(\tau_{i,j}) = max\big\{&T_s \cdot (j_s - 1) | \forall \tau_s \in V, \theta_s = \text{“sen”}, \\ &\forall j_s \in [1, n\text{-}ins(\tau_s, \Delta)], \quad \text{if } U_{(i,j),(s,j_s)} = 1\big\} \end{aligned} \tag{12}$$

Once we determine the OTS and NTS values for each sink instance, we can derive the reaction time and time disparity. To obtain the maximum values across all sink node instances within the interval Δ, OTS and NTS need to be calculated for each instance. We define MRT following the first-to-first concept from [8], and MTD according to the worst-case time disparity definition in [13]. Using our ILP model, these metrics for a sink node $\tau_\otimes$ are formulated as follows:

$$MRT(\tau_\otimes) = \max\{f_{\otimes,j} - OTS(\tau_{\otimes,j-1}) | \forall j \in [1, n\text{-}ins(\tau_\otimes, \Delta)]\} \tag{13}$$

$$MTD(\tau_{\otimes}) = \max\{NTS(\tau_{\otimes,j}) - OTS(\tau_{\otimes,j}) | \forall j \in [1, n\text{-}ins(\tau_{\otimes}, \Delta)]\} \tag{14}$$

To compute PAoI, we determine the time interval between the initiation of each sensor instance and its most recent update for every sensor contributing to a sink node. We then take the maximum interval across sensors per sink instance, and finally, the maximum among all sink instances. We formalize PAoI as follows, where s_{s,j_s} denotes the start time of the sensor instance τ_{s,j_s}:

$$\begin{aligned}&PAoI(\tau_{\otimes}) = max\{(s_{s,j_s} - s_{s,j_s-1}) \cdot U_{(\otimes,j),(s,j_s)} | \\ &\forall j_s \in [1, n\text{-}ins(\tau_s, \Delta)], \quad \forall \tau_s \in V, \quad \forall j \in [1, n\text{-}ins(\tau_{\otimes}, \Delta)]\}\end{aligned} \tag{15}$$

The metrics above are formulated for each sink node for the entire DAG. In addition, we can determine the worst-case response time (WCRT) of individual chains from each sensor τ_s to each sink node $\tau_{\otimes}$ as follows:

$$\begin{aligned}&WCRT(\tau_s, \tau_{\otimes}) = \max\{(f_{\otimes,j} - T_s \cdot (j_s - 1)) \cdot U_{(\otimes,j)(s,j_s)} | \\ &\forall j \in [1, n\text{-}ins(\tau_{\otimes}, \Delta)], \quad \forall j_s \in [1, n\text{-}ins(\tau_s, \Delta)]\}\end{aligned} \tag{16}$$

Note that, unlike MRT, MTD, and PAoI, which consider all sensors contributing to an actuator, WCRT is specified for a single sensor.

Another metric modeled in our work is the makespan (MS), often used to enhance resource efficiency. Makespan is the total time taken to complete a set of tasks, and in our model, it corresponds to the maximum finish time of a sink instance across all its instances. In ADS, however, where tasks can be triggered by events or on a timer, minimizing MS may seem less relevant since tasks cannot begin before their designated trigger times. Still, including MS as an optimization objective may provide insights for fine-tuning and calibrating the system. MS is formulated as follows for a sink node $\tau_{\otimes}$:

$$MS(\tau_{\otimes}) = \max\{f_{\otimes,j} | \forall j \in [1, n\text{-}ins(\tau_{\otimes}, \Delta)]\} \tag{17}$$

Determining Time Interval Δ. Recall that we define the time interval Δ as $\Delta = k \cdot HP$, where HP is the hyperperiod (Sec. 4). In conventional static schedule generation, $k = 1$ is sufficient to check task-level schedulability because the same schedule will repeat afterwards. However, for end-to-end metrics like MRT, the initial hyperperiod serves as a warm-up phase [31], where all tasks and paths within the DAG complete at least one execution cycle (e.g., ensuring data availability when T-fusion and I-fusion nodes are triggered subsequently). Therefore, in our ILP-based framework, we use $\Delta = 3 \cdot HP$ consisting of three phases: (i) first HP: a warm-up phase for the second HP and is excluded from performance evaluation; (ii) second HP: the optimized schedule that will repeat afterwards; (iii) third HP: an exact copy of the second HP's schedule with the same relative start and finish times for all task instances. The third HP is essential for evaluating end-to-end latency metrics (MRT, MTD, and PAoI), as data flows may span across hyperperiod boundaries. Hence, we optimize these metrics for two consecutive hyperperiods (the second and third) to capture cross-boundary data flows. Once an optimized schedule is determined, it can be applied to runtime systems by enforcing the predetermined start and finish times of the task instances and repeating the second hyperperiod's schedule.

With these metrics formulated by our ILP model, we define our ILP objective function to *minimize* these metrics using a hierarchical-blended approach [12], where metrics are first optimized using lexicographic ordering by priority, and within the same priority level, a weighted sum is applied. This approach provides configurable priorities and weights that let us choose which metric(s) to focus on. It is worth mentioning that while each metric is defined as a maximum (upper bound) value among different instances, our optimization objective is to minimize these maximum values.

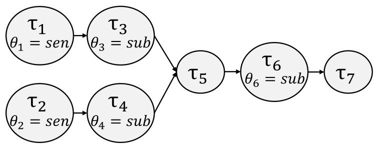
6 Evaluation

Our evaluation begins with case studies derived from existing literature, with extensions to capture additional fusion task types. We further evaluate our framework using randomly generated DAGs, varying the number of nodes, edges, and task types. The framework is implemented using Gurobi, a state-of-the-art ILP solver capable of efficiently handling linear, mixed-integer, and quadratic problems, and the optimal schedule from the framework has been tested on a Raspberry Pi. We also provide additional customized case studies in Appendix B.2 in [24], focusing on unique features of our framework to highlight how our model handles diverse DAG configurations. Throughout this section, the relative deadlines of timer-triggered tasks are set equal to their periods, while for event-triggered tasks, the relative deadline is set to the largest period among the timer-triggered tasks. Additionally, the multi-objective function in our framework includes all metrics—MRT, MTD, PAoI, WCRT, and MS—with weights and priorities set to 1, unless stated otherwise.

6.1 Case Studies from [32] and Beyond

We compare our ILP-based framework to the approach proposed in [32], which focuses on optimizing MRT for chains within DAGs while supporting only certain types of fusion nodes. Their method represents one of the most advanced existing models for cause-effect chains and is evaluated using two case studies. To ensure a fair comparison, we align our setup by configuring the number of CPU cores to $\Pi = 1$, matching their single-core assumption. We apply our work to the same case studies and expand the task configurations beyond their model's support.

Fusion System with Two Chains. To facilitate comparison, we adopted the DAG structure shown in Fig. 4a from [32] with minor adjustments. In [32], the authors explored two types of fusion nodes and two types of actuator nodes across four different configurations. In our setup, we mapped their "subscription-fusion" node type to our W-fusion node type; "timer-fusion" node to our T-fusion node; "timer actuator" node to our T-fusion sink node with a single predecessor; and "subscription-actuator" node to a subscription node in our model. In addition, to evaluate the MTD metric, we introduced a new configuration in which sensors have non-harmonic periods. Therefore, we used five configurations in total: *(i)* WS: $\underline{\text{W}}$-fusion node with $\underline{\text{S}}$ubscription actuator, *(ii)* WT: $\underline{\text{W}}$-fusion node with



τ_1 $\theta_1 = sen$
τ_3 $\theta_3 = sub$
τ_2 $\theta_2 = sen$
τ_4 $\theta_4 = sub$
τ_5
τ_6 $\theta_6 = sub$
τ_7

(a) DAG structure

(b) Task configurations

Config	θ_5	θ_7	Task periods (ms)
WS	"w-fus"	"sub"	$T_1 = T_2 = 360$
WT	"w-fus"	"t-fus"	$T_1 = T_2 = 420$, $T_7 = 840$
TS	"t-fus"	"sub"	$T_1 = T_2 = 420$, $T_5 = 840$
TT	"t-fus"	"t-fus"	$T_1 = T_2 = 480$, $T_5 = T_7 = 960$
NH	"t-fus"	"t-fus"	$T_1 = 480$, $T_2 = 360$, $T_5 = T_7 = 960$

Fig. 4: Fusion system with two chains

Table 2: MRT comparison with prior work and PAoI/WCRT results

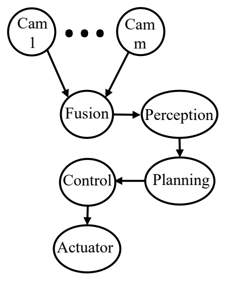
Config	**[32]'s UB**	**[32]'s LB**	**IDS:MRT**	**IDS:PAoI**	**IDS:WCRT of chain 1**
WS	2190	510	510	360	150
WT	3780	1320	990	120	150
TS	2420	1410	990	60	150
TT	3830	2490	1110	60	150
NH	3830	2490	1250	40	250

T-fusion actuator, *(iii)* TS: T-fusion node with Subscription actuator, *(iv)* TT: T-fusion node with T-fusion actuator, and *(v)* NH: Non-Harmonic periods for the TT configuration. The period T_i and type θ_i of tasks that vary across configurations are listed in Table 4b. For all configurations, we set the WCET of the tasks as follows: $e_1 = e_3 = 10(ms)$, $e_2 = e_4 = 20(ms)$, $e_5 = e_6 = e_7 = 30(ms)$.

Table 2 summarizes the comparison results, where IDS indicates the results from our framework (ILP-based DAG Schedule) when optimizing a multi-objective function of MRT, MTD, PAoI, and WCRT metrics. We include the analytical upper-bound (UB) and the simulation lower-bound (LB) on MRT computed using [32]'s implementation[2]; however, note that these are for reference only as [32] analyzes MRT bounds under ROS2 scheduling while our framework finds the performance bounds of various metrics through an optimal schedule. The results show that our optimal schedule can achieve substantially lower MRT compared to conventional scheduling while also optimizing other metrics. MTD is not reported in the table since it is zero for the first four configurations due to the same sensor periods, and equals 120 ms for the NH configuration.

Navigation System. This case study from [32], inspired by a navigation system, models its DAG as shown in Fig. 5a. The number of cameras can vary, allowing us to assess the impact of scaling on system performance. We denote the optimal result from our framework as IDS. We also report the MRT value obtained by implementing our optimal schedule as a static user-level scheduler in Linux and running it for 100 hyperperiods on a Raspberry Pi 4 (64-bit quad-core ARM Cortex-A72), denoted as the observed MRT. We then compare this against the maximum UB and LB on MRT across all chains computed by [32], as the number of cameras increases. To align with their model, we map their fusion node to our W-fusion type. We adopt the same WCET values as in [32] for all other nodes. Table 5b shows that our observed MRT matches their simulation LB at lower camera counts and stays lower when their MRT increases at 10 cameras. Results for other metrics are as follows: MTD = 0 and PAoI = 100 ms for all

[2] [32] analyzes only the MRT of each chain. Hence, we report the maximum MRT upper-bound and lower-bound across all chains in the DAG.



(a) Navigation system with m cameras

(b) MRT comparison between [32] (UB and LB) and our IDS output executed on Raspberry Pi 4

m	[32]'s UB	[32]'s LB	IDS:MRT	observed MRT (ms)
1	480	155	155	155.06
2	515	160	160	159.85
3	840	165	165	164.94
4	890	170	170	170.37
5	940	175	175	175.38
6	990	180	180	180.31
7	1040	185	185	184.94
8	1090	190	190	190.48
9	1140	195	195	194.93
10	1190	530	200	200.01

Fig. 5: Navigation System Case Study

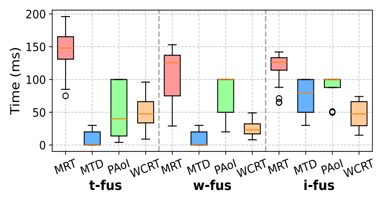
camera counts (due to fixed camera periods); MS increases from 255 ms to 300 ms; and WCRT increases from 55 ms to 100 ms, both rising in 5 ms increments per additional camera. We introduce additional configurations beyond [32]'s case study, where the fusion node is a T-fusion or I-fusion. Due to space limits, the results are provided in our extended manuscript [24] (Appendix B.1).

6.2 Randomly-Generated DAGs

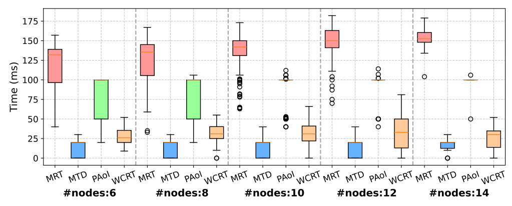
To evaluate our framework with more complex DAGs with diverse fusion types, we use randomly generated DAGs. We conducted this set of experiments on a Linux server equipped with two AMD EPYC 7452 processors (32 cores each), providing a total of 64 cores and 256GB (16GB x 16) of DDR4 RAM.

The size of each experiment is set to 100 randomly generated DAGs. The WCET of event-triggered tasks was randomly selected from $[1, 5](ms)$. For timer-triggered tasks, we assigned a random utilization from $[0.1, 0.4]$ and chose a period from $\{20, 40, 50, 100\}(ms)$, calculating their WCET as the product of utilization and period. The number of cores used for scheduling is set to $\Pi = 2$. Since our model supports multiple sink nodes, we calculated MRT, MTD, and PAoI for the sink with the last index. For WCRT, defined along a chain from a sensor to a sink node, we focused on the path between the sensor with the first index and the sink with the last index.

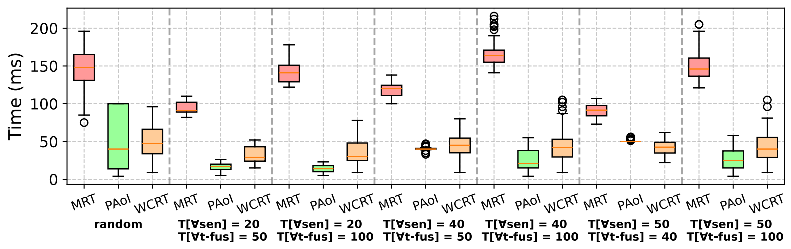
We first explored the impact of fusion types on a set of 100 DAGs, each consisting of 6 nodes — 3 of which are sensor nodes — and 7 edges. Using these same DAG structures, we varied the fusion node type by allowing multiple fusion nodes, but all of the same type — either T-fusion, W-fusion, or I-fusion — within each DAG. The distribution of optimized metrics across the feasible cases (out of 100) is shown in Fig. 6a, highlighting the expected effects of each fusion type. For example, comparing MRT and MTD across fusion types shows that I-fusion tends to reduce MRT by acting on each new input immediately. However, this same behavior increases MTD as it leads to multiple reuses of some inputs, increasing the deviation between the oldest and newest sensor data used for an actuator. We can also observe that T-fusion may result in high MRT



(a) Varying the allowed fusion types



(b) Varying #nodes and #edges (W-fusion allowed)

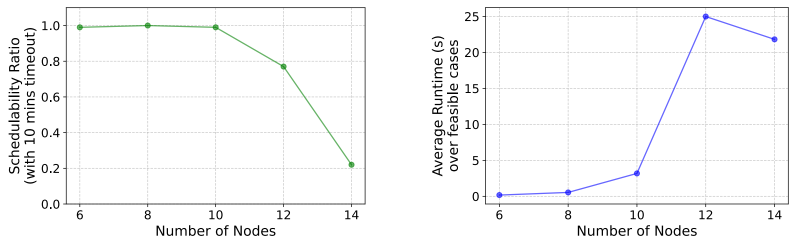


(c) Varying the periods (T-fusion allowed)

Fig. 6: Distribution of optimized real-time performance metrics across feasible cases

if task periods are not carefully chosen. To better investigate how task periods affect these metrics, we conducted another experiment in which only T-fusion is allowed for fusion nodes, assuming task periods can be configured at design time, as shown in Fig. 6c. We compared the default case where sensor and T-fusion node periods are randomly selected from {20, 40, 50, 100}(ms) to cases with fixed period assignments. For simplicity, we assume all sensors (and T-fusion nodes) share the same period, denoted as $T[\forall sen]$ (and $T[\forall t\text{-}fus]$) in the figure. We excluded MTD from the results since it is zero in all cases except the random one, due to uniform sensor and T-fusion periods. The results show that there exist cases where the random case has better MRT than those with T-fusion periods much shorter than sensor periods. In fact, MRT improves when sensor and T-fusion periods are closely aligned. We also found that lower PAoI occurs when sensors have shorter periods and sample data more frequently. In this experiment, we can see that if periods can be tuned, then our framework gives valuable insight on their effect on real-time metrics.

In another experiment, we evaluated our framework by increasing DAG complexity through varying the number of nodes and edges. We considered five configurations, assuming the number of sensor nodes to be half the total nodes and the number of edges to be twice the nodes. The number of edges was kept within the valid DAG range, $[|V|-1, |V|\cdot(|V|-1)/2]$ where $|V| = m + n$ is the total number of nodes. For each configuration, we generated 100 DAGs and analyzed the distribution of real-time metrics, as shown in Fig. 6b. In this experiment, we fixed the fusion type to W-fusion, except for I-fusion following branch nodes. The results show that real-time metrics, such as MRT, increase with DAG complexity, while the number of feasible cases declines due to the growing workload on a fixed number of cores ($\Pi = 2$). We further evaluated the schedulability ratio and the average runtime of our framework using the same configurations. The schedulability ratio is the ratio of feasible (schedulable) cases returned by



(a) Schedulability ratio (b) Average runtime

Fig. 7: The schedulability ratio (10-minute timeout) and average runtime per case of our framework

our framework within 10 minutes out of 100 DAGs. As shown in Fig. 7, both the schedulability ratio and the average runtime for feasible cases are influenced by the increasing complexity of the DAGs, while the number of resources remains unchanged. Note that the smaller runtime observed for 14 nodes compared to 12 nodes in Fig. 7b, as well as the lower MTD and PAoI values for 14 nodes than those for 12 nodes in Fig. 6b, are both due to the fewer feasible cases.

To further assess the flexibility of our framework, we increased the number of cores, which enabled scheduling even more complex DAGs with improved success rates and scalable average runtimes. The results of this extended evaluation are provided in [24] (Appendix B.3) due to space limits.

7 Conclusion

In this paper, we introduce flexible and structured modeling for data fusion tasks in the Autonomous Driving System (ADS) software stack, supporting complex task types and chains with diverse triggering options that existing models have not addressed comprehensively and systematically. We present an ILP-based framework that quantitatively compares different fusion patterns and their impact on real-time performance metrics, providing optimized resource allocation (task-to-core mapping) and timing schemes for all task instances. Our framework can optimize various real-time performance metrics, such as Maximum Reaction Time (MRT), Maximum Time Disparity (MTD), Peak Age of Information (PAoI), Worst-Case Response Time (WCRT), and Makespan (MS), allowing users to adjust fusion strategies and other system parameters. Evaluation against existing approaches demonstrates that our framework not only better handles complex DAG structures found in real systems, but also effectively analyzes achievable bounds for key performance metrics. In the future, we plan to improve the scalability of our framework by incorporating heuristics such as simulated annealing and learning-based techniques for highly complex ADS.

Acknowledgment

This work was sponsored by the National Science Foundation (NSF) grants 1943265 and 2312395.

References

1. Abdullah, J., Dai, G., Yi, W.: Worst-case cause-effect reaction latency in systems with non-blocking communication. In: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1625–1630. IEEE (2019)
2. Autoware Foundation: Autoware (2024), `https://autoware.org`
3. Becker, M., Dasari, D., Mubeen, S., Behnam, M., Nolte, T.: Synthesizing job-level dependencies for automotive multi-rate effect chains. In: 2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). pp. 159–169. IEEE (2016)
4. Becker, M., Mubeen, S., Dasari, D., Behnam, M., Nolte, T.: A generic framework facilitating early analysis of data propagation delays in multi-rate systems. In: 2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). pp. 1–11. IEEE (2017)
5. Cao, A., Shen, C., Zong, J., Chang, T.H.: Peak age-of-information minimization of uav-aided relay transmission. In: 2020 IEEE International Conference on Communications Workshops (ICC Workshops). pp. 1–6. IEEE (2020)
6. Choi, H., Karimi, M., Kim, H.: Chain-based fixed-priority scheduling of loosely-dependent tasks. In: 2020 IEEE 38th International Conference on Computer Design (ICCD). pp. 631–639. IEEE (2020)
7. Davies, A., Poorswani, N., Xiang, R., Brown, B., Singh, R., Gupta, S., Vafaee, A., Han, J.H., Tumati, P., Janapareddy, S., Tadkase, A.: Brief Industry Paper: STM: A Static Non-preemptive Scheduler for NVIDIA Tegra SoC. In: 2025 IEEE 31st Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE (2025)
8. Dürr, M., Brüggen, G.V.D., Chen, K.H., Chen, J.J.: End-to-end timing analysis of sporadic cause-effect chains in distributed systems. ACM Transactions on Embedded Computing Systems (TECS) **18**(5s), 1–24 (2019)
9. Günzel, M., Chen, K.H., Ueter, N., von der Brüggen, G., Dürr, M., Chen, J.J.: Timing analysis of asynchronized distributed cause-effect chains. In: 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS). pp. 40–52. IEEE (2021)
10. Günzel, M., Chen, K.H., Ueter, N., Brüggen, G.v.d., Dürr, M., Chen, J.J.: Compositional timing analysis of asynchronized distributed cause-effect chains. ACM Transactions on Embedded Computing Systems **22**(4), 1–34 (2023)
11. Günzel, M., Teper, H., Chen, K.H., von der Brüggen, G., Chen, J.J.: On the equivalence of maximum reaction time and maximum data age for cause-effect chains. In: 35th Euromicro Conference on Real-Time Systems (ECRTS 2023). Schloss-Dagstuhl-Leibniz Zentrum für Informatik (2023)
12. Gurobi Optimization, LLC: Combining blended and hierarchical objectives (2025), `https://docs.gurobi.com/projects/optimizer/en/current/features/multiobjective.html#combining-blended-and-hierarchical-objectives`
13. Jiang, X., Luo, X., Guan, N., Dong, Z., Liu, S., Yi, W.: Analysis and optimization of worst-case time disparity in cause-effect chains. In: 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1–6. IEEE (2023)
14. Kim, J., Kim, H., Lakshmanan, K., Rajkumar, R.: Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car. In: Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems (ICCPS) (2013)

15. Kuhse, D., Holscher, N., Gunzel, M., Teper, H., Von Der Bruggen, G., Chen, J.J., Lin, C.C.: Sync or sink? the robustness of sensor fusion against temporal misalignment. In: 2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS). pp. 122–134. IEEE (2024)
16. Li, R., Jiang, X., Dong, Z., Wu, J.M., Xue, C.J., Guan, N.: Worst-case latency analysis of message synchronization in ros. In: 2023 IEEE Real-Time Systems Symposium (RTSS). pp. 185–197. IEEE (2023)
17. Li, X., Ma, Y., Chen, Y., Sun, J., Chang, W., Guan, N., Chen, L., Deng, Q.: Priority optimization for autonomous driving systems to meet end-to-end latency constraints. In: 2024 IEEE Real-Time Systems Symposium (RTSS). pp. 402–414. IEEE (2024)
18. Maia, L., Fohler, G.: Reducing end-to-end latencies of multi-rate cause-effect chains in safety critical embedded systems. In: 12th European Congress on Embedded Real Time Software and Systems (ERTS 2024) (2024)
19. PerceptIn: RTSS 2021 industry challenge (2021), `https://2021.rtss.org/wp-content/uploads/2021/06/RTSS2021-Industry-Challenge-v2.pdf`, accessed: 2025-02-23
20. Saidi, S.E., Pernet, N., Sorel, Y.: Automatic parallelization of multi-rate fmi-based co-simulation on multi-core. In: TMS/DEVS 2017-Symposium on Theory of Modeling and Simulation. pp. Article–No. ACM (2017)
21. Saito, Y., Azumi, T., Kato, S., Nishio, N.: Priority and synchronization support for ros. In: 2016 IEEE 4th International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA). pp. 77–82. IEEE (2016)
22. Saito, Y., Sato, F., Azumi, T., Kato, S., Nishio, N.: Rosch: real-time scheduling framework for ros. In: 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). pp. 52–58. IEEE (2018)
23. Sobhani, H., Choi, H., Kim, H.: Timing analysis and priority-driven enhancements of ros 2 multi-threaded executors. In: 2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS). pp. 106–118. IEEE (2023)
24. Sobhani, H., Kim, H.: Modeling and scheduling of fusion patterns in ads (extended version) (2025), `https://doi.org/10.48550/arXiv.2510.23895`
25. Sun, J., Duan, K., Li, X., Guan, N., Guo, Z., Deng, Q., Tan, G.: Real-time scheduling of autonomous driving system with guaranteed timing correctness. In: 2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS). pp. 185–197. IEEE (2023)
26. Sun, J., Li, X., Gong, M., Guan, N., Guo, Z., Chen, M., Zhao, J., Deng, Q.: Jointly ensuring timing disparity and end-to-end latency constraints in hybrid dags. In: 2025 IEEE 31st Real-Time and Embedded Technology and Applications Symposium (RTAS). pp. 190–201. IEEE (2025)
27. Sun, J., Wang, T., Li, Y., Guan, N., Guo, Z., Tan, G.: Seam: An optimal message synchronizer in ros with well-bounded time disparity. In: 2023 IEEE Real-Time Systems Symposium (RTSS). pp. 172–184. IEEE (2023)
28. Tang, Y., Guan, N., Jiang, X., Dong, Z., Yi, W.: Reaction time analysis of event-triggered processing chains with data refreshing. In: 2023 60th ACM/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2023)
29. Tang, Y., Jiang, X., Guan, N., Ji, D., Luo, X., Yi, W.: Comparing communication paradigms in cause-effect chains. IEEE Transactions on Computers **72**(1), 82–96 (2022)

30. Tang, Y., Jiang, X., Guan, N., Liu, S., Luo, X., Yi, W.: Optimizing end-to-end latency of sporadic cause-effect chains using priority inheritance. In: 2023 IEEE Real-Time Systems Symposium (RTSS). pp. 411–422. IEEE (2023)
31. Teper, H., Betz, T., Günzel, M., Ebner, D., Von Der Brüggen, G., Betz, J., Chen, J.J.: End-to-end timing analysis and optimization of multi-executor ros 2 systems. In: 2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS). pp. 212–224. IEEE (2024)
32. Teper, H., Günzel, M., Ueter, N., von der Brüggen, G., Chen, J.J.: End-to-end timing analysis in ros2. In: 2022 IEEE Real-Time Systems Symposium (RTSS). pp. 53–65. IEEE (2022)
33. Toba, H., Azumi, T.: Deadline miss early detection method for dag tasks considering variable execution time. In: 36th Euromicro Conference on Real-Time Systems (ECRTS 2024). Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2024)
34. Verucchi, M., Theile, M., Caccamo, M., Bertogna, M.: Latency-aware generation of single-rate dags from multi-rate task sets. In: 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). pp. 226–238. IEEE (2020)
35. Wang, S., Chen, M., Yang, Z., Yin, C., Saad, W., Cui, S., Poor, H.V.: Distributed reinforcement learning for age of information minimization in real-time iot systems. IEEE Journal of Selected Topics in Signal Processing **16**(3), 501–515 (2022)
36. Xu, C., Xu, Q., Wang, J., Wu, K., Lu, K., Qiao, C.: Aoi-centric task scheduling for autonomous driving systems. In: IEEE INFOCOM 2022-IEEE Conference on Computer Communications. pp. 1019–1028. IEEE (2022)
37. Yano, A., Azumi, T.: Deadline miss early detection method for mixed timer-driven and event-driven dag tasks. IEEE Access **11**, 22187–22200 (2023)
38. Yano, A., Azumi, T.: Work-in-progress: Multi-deadline dag scheduling model for autonomous driving systems. In: 2024 IEEE Real-Time Systems Symposium (RTSS). pp. 451–454. IEEE (2024)
39. Zhu, Q., Li, W., Kim, H., Xiang, Y., Wardega, K., Wang, Z., Wang, Y., Liang, H., Huang, C., Fan, J., Choi, H.: Know the unknowns: Addressing disturbances and uncertainties in autonomous systems. In: International Conference on Computer-Aided Design (ICCAD) (2020)