Toward the Right Analytical Model and System Software for Autonomous Driving Systems: Open Problems and Research Directions

Atsushi Yano*† and Takuya Azumi‡†
*Graduate School of Science and Engineering, Saitama University, Japan
†TIER IV Incorporated, Japan
‡Academic Association (Graduate School of Science and Engineering), Saitama University, Japan

***Abstract*—Autonomous driving (AD) systems continuously transform multi-rate and asynchronous sensor streams into vehicle actuation through graphs of callbacks, nodes, and middleware components. In such systems, temporal correctness cannot be characterized by the execution time or deadline of an individual task alone: localization and perception chains run in parallel, fuse data with different timestamps, converge at planning, and propagate through control to actuation. Moreover, the demand for high processing capability places AD systems on high-performance processors with multicore parallelism and GPU acceleration, where execution times vary strongly with the input scene, hardware state, and co-running work. Rare deadline misses at runtime therefore cannot be ruled out, and safety is preserved through fail-safe mechanisms such as the minimal-risk maneuver (MRM). This raises a two-sided question: what analytical models are needed to reason about timing in AD systems, and what system software is needed to realize, observe, and enforce those models on real platforms? On the analytical side, real-time research has evolved from periodic/sporadic tasks, directed acyclic graphs (DAGs), pipelines, mixed-criticality systems, and timer-/event-driven models toward end-to-end latency along cause-effect chains, data freshness, timing disparity, probabilistic timing, highest-criticality fail-safe operation, and early deadline-miss detection. On the system-software side, AD stacks and middleware, such as Autoware and ROS 2, expose both the opportunities and limitations of implementing analyzable timing behavior through executors, communication layers, tracing tools, and evaluation frameworks. This paper surveys these two lines of work and identifies the remaining gaps along five dimensions: units of timing constraints, timing metrics, resource models, execution-time variability, and safety integration. Rather than proposing a single new model or runtime, we formulate open problems and research directions for converging theory and practice: analytical models must move closer to AD reality, while AD system software must be reshaped into analyzable, enforceable, and safety-aware infrastructure.**

***Index Terms*—real-time systems, autonomous driving, analytical timing model, system software, ROS 2, Autoware, cause-effect chains, end-to-end latency, data freshness, probabilistic timing, deadline-miss detection, open problems.**

I. Introduction

An autonomous driving (AD) system is a cyber-physical system that continuously transforms multiple sensor streams into vehicle actuation through a graph of callbacks rather than a single linear chain (Fig. 1). This figure shows one representative Autoware configuration, a concrete instance rather than a fixed architecture. In it, each gray box is a Robot Operating System (ROS) 2 callback (or the node itself when the node holds a single callback), a dashed enclosure groups the callbacks of a multi-callback node, and solid edges carry publish/subscribe communication over a topic. Orange dotted edges instead carry data through a node-internal queue or the ROS 2 take API [1]; such an edge does not trigger the destination callback directly but affects its execution time and output. Each callback is triggered by a timer at a fixed period (marked with a timer icon), by a subscription to a single topic, or by the synchronization of several topics (a sync callback, using the ApproximateTime or ExactTime policy [2]). Multi-rate sensors, such as multiple LiDARs, a global navigation satellite system (GNSS), and an inertial measurement unit (IMU), feed two parallel chains: a localization chain (ndt scan matcher and ekf localizer, constrained by a 20 ms deadline) and a perception chain (GPU-based pointpainting, object detection, and tracking driven by a 100 ms timer). These chains converge at planning and propagate through control (a 30 ms timer) to actuation. In addition, some of these nodes, such as ekf localizer, planning, and control, are replicated for the minimal-risk maneuver (MRM), an emergency stop based on dead reckoning, and the replicas reside both on the same host and on a redundant electronic control unit (ECU) [3]. In such a graph, it is not sufficient that each node eventually produces a correct result. The sensor data must not become too stale, the timestamps of data fused from different sensors must lie within a tolerable window, and a control command must be produced within a time that is still physically meaningful for the moving vehicle. Timing correctness is therefore a property of the whole sensor-to-actuation graph rather than of any single task.

Classical real-time systems theory analyzes the schedulability of task sets built on periodic and sporadic tasks, worst-case execution time (WCET) and worst-case response time (WCRT), deadlines, and fixed-priority or earliest-deadline-first (EDF) scheduling [4], [5]. In AD software, these assumptions are repeatedly strained. (i) The software is not a single task but many callbacks and nodes forming a graph. (ii) Sensor inputs and events are not co-periodic; the system is simultaneously multi-rate and event-driven. (iii) Because AD systems demand

high processing capability, they run on modern processors with rich acceleration features, such as caches, speculative execution, multicore parallelism, and offloading to GPUs and neural processing units (NPUs). On these platforms, CPU, GPU, NPU, communication, and memory contend in complex ways. (iv) Execution time varies strongly not only with the input scene and data volume but also with the hardware state and co-running work. A single fixed WCET bound therefore becomes overly pessimistic, while a safety margin added to measured maxima can underestimate the true worst case. (v) Because absolute timing guarantees are impractical under such variability, rare deadline misses at runtime cannot be ruled out, and AD systems preserve safety against them through fail-safe mechanisms such as fallback, degraded modes, and the MRM [3]. A timing violation must therefore be interpreted not merely as a deadline miss but together with these safety mechanisms.

This paper asks what the right *analytical model* and *system software* for AD systems should be, and organizes the answer from two directions. Section II surveys the analytical timing models and identifies five concrete gaps between classical assumptions and AD reality (Table I). Section III maps the implementation and evaluation research, centered on Autoware and ROS 2, that both makes those models observable and controllable on real systems and reveals where real systems break the assumptions theory relies on (Table II). Sections IV and V then pose the open problems that remain as two parallel lists: open problems for the real-time *theory* community, and open problems for *practitioners* who build and deploy AD software, respectively. Our central argument is that this gap cannot be closed from either side alone: the theory must evolve toward AD reality, and AD implementations must in turn be reshaped so that they faithfully realize analyzable models. Our contribution is therefore not a single new task model but (i) a structured account of which existing models must be combined and where the theory–implementation gap is still open, and (ii) a pair of open-problem agendas, one for each community, that together define this two-way convergence.

II. Analytical Models and the Theory–Reality Gap

A. Timing Requirements in Real AD Systems

Six timing requirements recur in AD systems, and a single deployed AD system must satisfy all of them at once.

(R1) End-to-end (E2E) latency along cause-effect chains: the total delay from sensor input to actuator output must be bounded even when every individual node meets its deadline [7]–[10]. Bounding this delay requires tracing which input sample influences which output decision as a sensor reading propagates through perception into planning and control [7], [11], [12].

(R2) Response time and schedulability: the classical question of whether a unit of work activated at a single rate, such as a sporadic or periodic task, a callback chain, or a directed acyclic graph (DAG), completes within its deadline, now posed on top of middleware executors [13]–[15].

(R3) Data freshness and timing disparity: the age of data and the timestamp gap between fused sensor streams must be controlled; for example, LiDAR and camera results do not necessarily describe the environment at the same instant [16]–[18].

(R4) Probabilistic timing: because a fixed WCET is overly pessimistic while the average case is unsafe, timing should be expressed probabilistically, for example as the exceedance probability of the reaction time given input execution-time distributions [19], or as probabilistic guarantees on E2E latency [20]. Probabilistic WCET (pWCET) makes this concrete by upper-bounding the probability that the execution time exceeds a given value, so that a timing bound can be matched to the failure rate tolerated by the safety goal [21], [22].

(R5) Highest-criticality runtime fail-safe: because the rare misses that such probabilistic bounds tolerate can still occur at runtime, a fail-safe mechanism such as the MRM must be provisioned to preserve safety when they do. As the last line of defense, it must remain operational precisely when the nominal functions are failing; therefore, its execution must be guaranteed at the highest criticality [23].

(R6) Deadline-miss early detection: the possibility of a miss should be detected *during* execution, early enough to trigger such a safe transition rather than after the fact [24], [25].

Recent frameworks make several of these metrics, notably maximum reaction time (MRT) and maximum data age (MDA), first-class, both in ROS 2 [8], [17] and in CyberRT/Apollo [26].

B. Existing Models and Their Assumptions

None of these requirements is uncharted: each is attacked by an active line of work built from a small set of building blocks. For E2E latency (R1), cause-effect-chain analysis bounds reaction time and data age over ROS 2 callback graphs [8], and E2E latency is measured and optimized for containerized ROS 2 AD software [10]. For response time (R2), the periodic/sporadic task model and its response-time analysis are re-instantiated on middleware executors via chain-aware priorities [13], multi-threaded executor analysis [14], and executor designs that restore the applicability of classical scheduling theory [15]. DAG and precedence models expose intra-task parallelism and dependencies [11], [12], and timer-driven/event-driven activation models capture how nodes are released [8], [18]. For freshness and disparity (R3), synchronization and buffering policies are modeled to bound worst-case time disparity [16], data age under refreshing [17], and disparity jointly with E2E latency [18]. For probabilistic timing (R4), pWCET frameworks [21], [22] and probabilistic analyses of reaction time [19] and E2E latency [20] replace single worst-case bounds with exceedance probabilities. For the highest-criticality fail-safe (R5), mixed-criticality scheduling co-hosts functions of different safety importance and preserves the guarantees of the most critical ones when optimistic execution-time budgets are exceeded [23]. For deadline-miss early detection (R6), runtime checks of DAG progress against

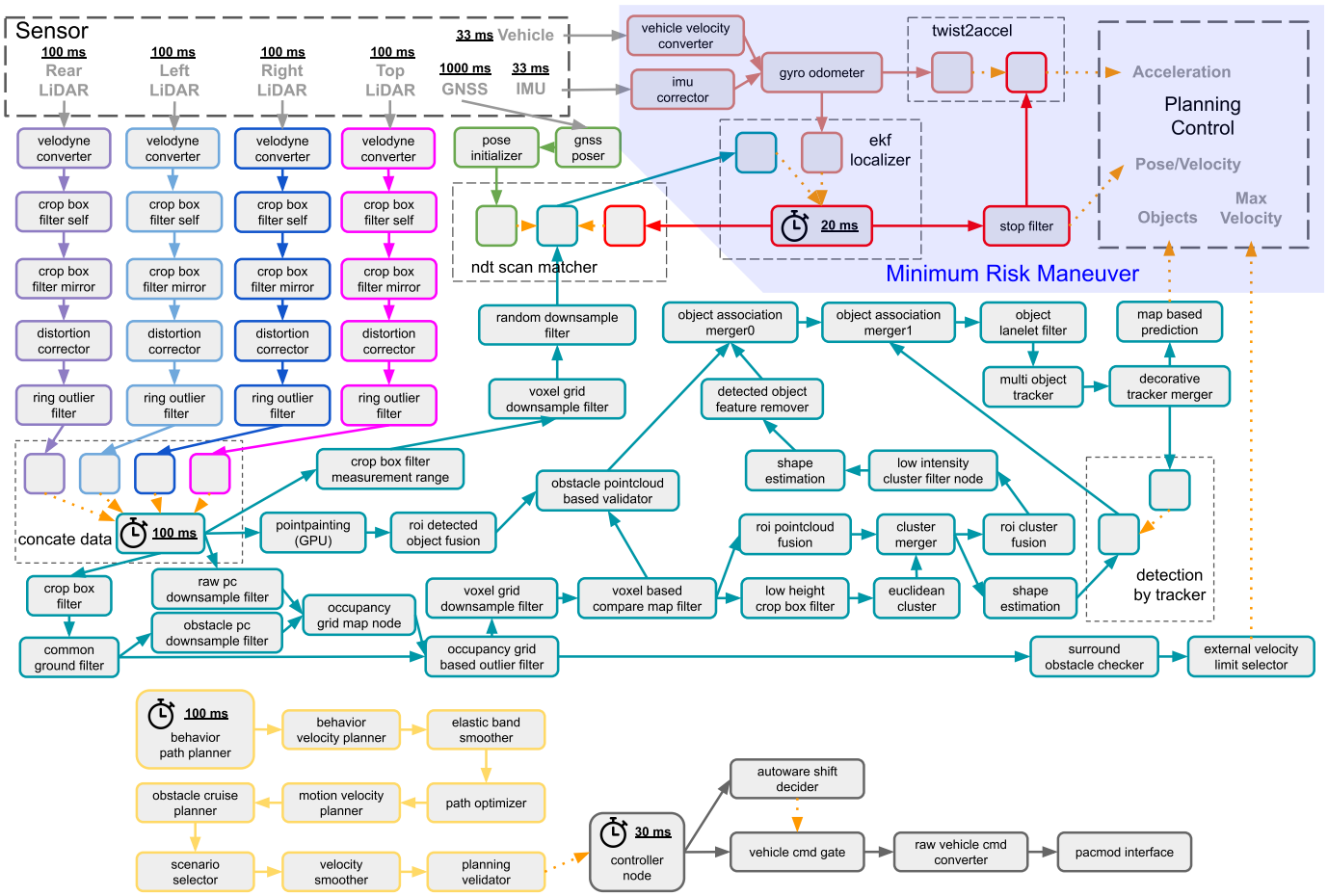

Fig. 1. Main callbacks and their trigger mechanisms in Autoware [6]. Sensor callbacks trigger two parallel chains: Localization (ndt scan matcher and ekf localizer with a 20 ms deadline) and Perception (GPU-based pointpainting, object detection, and tracking driven by a 100 ms timer). Both chains converge at Planning and proceed to Actuation via Control. The sensor pipeline required by ekf localizer for dead reckoning, ekf localizer itself, and parts of the subsequent localization, planning, and control are replicated for the minimal-risk maneuver (MRM) [3].

intermediate thresholds detect an impending miss during execution [24], [25].

These models are indispensable, but each line of work addresses its requirement largely in isolation, and how to combine them into one analysis that covers all six requirements for the same system remains open. Moreover, they are usually instantiated under assumptions that hold poorly for AD software: WCET is fairly well known, and computing resources other than the CPU are either not modeled or abstracted away.

C. Five Gaps

These assumptions diverge from AD reality along five dimensions, summarized in Table I.

(G1) Unit of timing constraint: classical analysis imposes the deadline on the same clear unit task that the scheduler dispatches, whereas in ROS 2/Autoware the schedulable unit is the callback while safety-relevant constraints such as E2E latency attach to an E2E cause-effect chain or the whole graph. Which unit (node, callback, chain, E2E cause-effect chain, or graph) should carry a given timing constraint is therefore not obvious.

(G2) Metric integration: deadlines alone are insufficient when age, freshness, jitter, disparity, and E2E latency matter at once.

(G3) Resource modeling: CPU-only models cannot express GPU/NPU inference, memory bandwidth, and many-core communication [27].

(G4) Execution-time variability: execution time depends strongly on the number of objects, point-cloud density, and scene complexity; thus, a single worst case or average is rarely actionable. Contention for shared resources on multicore and heterogeneous systems-on-chip (SoCs) adds hardware-induced variability [28]–[31], motivating pWCET timing models [21], [22].

(G5) Safety integration: safety assurance is meaningful only in connection with redundancy, degraded modes, and the MRM. It must therefore extend to runtime: hazardous states such as an impending deadline miss must be detected during execution, and the fail-safe mechanism triggered in response must be guaranteed to execute.

TABLE I
WHERE CLASSICAL REAL-TIME ASSUMPTIONS DIVERGE FROM AD REALITY.

Aspect	Classical real-time model	AD reality
Unit of timing constraint	deadline on a clear unit task	constraints at chain/graph level, scheduling at callback level; units do not coincide
Timing metric	deadline-centric (period, response time)	age, freshness, jitter, disparity, E2E latency
Resource	CPU-centric	CPU/GPU/NPU, communication, memory, IPC contention
Execution-time variability	small / bounded	scene-/data-dependent *and* hardware-induced (shared-resource contention), large
Safety	external, post-hoc	redundancy, degraded mode, MRM; runtime hazard detection, fail-safe execution

D. Requirements of a Practical Real-Time Task and Timing Model

The practical model is therefore not one new task model but a layered combination of the existing building blocks. This combination (a) takes the callback chain/graph, not a single task, as the unit of analysis; (b) integrates multi-rate and event-driven activation; (c) evaluates E2E latency together with freshness, jitter, and timing disparity; (d) incorporates heterogeneous CPU/GPU/NPU, communication, and memory contention; (e) uses variability-aware execution-time models that reflect artificial intelligence (AI) workloads, scene changes, and platform-level resource contention; and (f) connects to safety control via criticality, fallback, and the MRM, including runtime detection of impending deadline misses and guaranteed execution of the triggered fail-safe mechanism.

III. IMPLEMENTATION AND EVALUATION LANDSCAPE

The gap of Section II is not only that theory abstracts away AD reality; the implementation side contributes its own half, since real AD stacks are not built to be analyzed. Table II organizes the implementation and evaluation research along five layers, summarized below.

Application. Autoware is an open-source AD stack where abstract DAGs, pipelines, and cause-effect chains materialize as concrete callbacks, nodes, topics, and executors [32]. It is increasingly the common benchmark for AD timing studies [51]–[57].

Executor. In ROS 2, the executor connects a scheduling policy to actual callback dispatch. Existing work restructures the dispatch mechanism itself (CallbackGroup [33], thread-per-callback [34], and one-to-one callback-to-thread enforcement [36]), imposes scheduling policies on it (ROSCH [37], chain-aware priorities [13], and flexible OS-level scheduling [35]), or analyzes the resulting timing (multi-threaded executors [14] and reconciliation with classical periodic-task scheduling [15]). Together they show that the executor is itself part of the timing model rather than a transparent layer the analysis can ignore.

Communication. Data movement matters as much as scheduling: an early ROS 2 study quantified Data Distribution Service (DDS) transport latency, throughput, and memory overhead across vendors and quality-of-service (QoS) settings [38], and many-core platforms move inter-node communication onto a network-on-chip (NoC) [27], [39]. A shared-memory and zero-copy line of work (TZC [40], iceoryx [41], and Agnocast [42]) removes serialization and copy costs from inter-process communication (IPC), while CROS-RT aligns priorities across the application, middleware, and kernel layers to bound IPC response times [43].

Tracing & evaluation. To observe timing on real systems, ros2_tracing [44] (extended with message-flow analysis [58]), CARET [45], and Autoware_Perf [47] measure E2E and path-level latency, while TILDE additionally tracks per-message data freshness and detects deadline misses at runtime [46]. For controlled experiments, RD-Gen generates reproducible multi-rate DAG workloads [48].

OS & hardware. Finally, behavior is shaped by the platform: Autoware on Board runs a full AD stack on Linux-based embedded GPU devices [32], ROS-lite and ROS-lite2 port the ROS execution model onto real-time operating system (RTOS)-based many-core platforms [27], [39], and heterogeneous SoCs host such stacks under power and real-time constraints [49]. Toward software-defined vehicles, hypervisor- and container-based deployment of Autoware achieves near-bare-metal performance except for disk I/O [50].

Each surveyed effort strengthens timing at a single layer, but no combination of them yet yields a stack whose whole-system safety can be fully guaranteed. Taken together, Sections II and III show that the theory–reality gap has two facing edges: models that abstract away AD reality, and implementations that remain hard to analyze. Closing the gap is therefore a problem of *mutual convergence*: theory must move toward AD reality while implementations move toward analyzable structure, and neither community can close it alone. We make this concrete as two parallel sets of open problems: Section IV asks what *model* the theory community still owes, and Section V asks what *system* practitioners must build to meet it.

IV. OPEN PROBLEMS: FOR THEORISTS

The first set of open problems targets the real-time *theory* community. Building on the five gaps of Table I and the requirements of Section II, each problem below asks what new

TABLE II
IMPLEMENTATION AND EVALUATION LANDSCAPE FOR AD REAL-TIME SYSTEMS, CENTERED ON AUTOWARE AND ROS 2.

Layer	Representative artifacts	Timing aspect addressed
Application	Autoware [32]	realistic chains / nodes / topics
Executor	CallbackGroup executor [33], ThreadedCallback [34], PiCAS [13], multi-threaded executor timing [14], classical real-time reconciliation [15], ROSRT [35], callback enforcement [36], and ROSCH [37]	callback dispatch, chain priority, and response time
Communication	ROS 2/DDS transport evaluation [38], NoC communication [27], [39], TZC [40], iceoryx [41], Agnocast [42], and CROS-RT [43]	IPC / zero-copy, NoC, memory movement, and cross-layer priority
Tracing & evaluation	ros2_tracing [44], CARET [45], TILDE [46], Autoware_Perf [47], and RD-Gen [48]	latency tracing, freshness / deadline-miss detection, and reproducible multi-rate DAG benchmarks
OS & hardware	Autoware on Board [32], ROS-lite [27] and ROS-lite2 [39], heterogeneous SoC [49], and virtualized / containerized execution [50]	OS scheduling, affinity, bandwidth, and virtualization overhead

task or timing *model*, rather than what new implementation, is still missing.

(TP1) The right unit of analysis: What should be the unit of analysis: callback, node, chain, E2E cause-effect chain, or graph? Classical schedulability assumes a clear unit task, but in ROS 2 the schedulable unit (the callback) and the safety-relevant unit (the cause-effect chain or graph) do not coincide [13], [15]. Chain- and DAG-level models exist [6], [11], yet a single model whose unit simultaneously carries schedulability and safety meaning is still open. Theorists and practitioners need to converge, through dialogue, on a unit that is flexible enough to express the implementation structures that AD systems essentially require, yet faithful to the timing-constraint demands of the real system.

(TP2) Joint analysis of heterogeneous timing metrics: How can one analysis reason jointly about E2E latency, response time, freshness, timing disparity, and miss probability? These metrics carry different safety meanings, and only a few pairs are analyzed together today: disparity with E2E latency [16], [18] and MRT with MDA [17], [26]. Collapsing them into one number loses information; an analytical model that treats the full set jointly remains open. One direction is a vector-valued timing contract that constrains the metrics jointly, e.g., bounding freshness and disparity while optimizing E2E latency, so that their interactions are analyzed rather than averaged away.

(TP3) Heterogeneous resource models: How can a timing model explicitly capture GPU/NPU execution, memory bandwidth, and on-chip interconnect, rather than leaving them unmodeled or abstracted away? NoC-based and clustered many-core ports of the ROS execution model [27], [39], and heterogeneous SoCs [49] show that these resources are inside the timing budget; an analysis of ROS 2 systems that makes them first-class is open.

(TP4) Execution-time models under scene-/data-dependent and hardware-induced variability: How should execution time be modeled when it depends on scene, data volume, and deep neural network (DNN) pre-/post-processing? A single worst case or average is rarely actionable, and scene-dependent workload should be separated from hardware- and runtime-induced residual variation. The former is largely fixed once the scene is given (e.g., the workload scales with the number of detected objects or input points); thus, subsuming it into a probability distribution mischaracterizes it. Probabilistic timing [19], [20] is a starting point; a model that captures the scene-dependent workload as an explicit function of the scene and only the residual variation probabilistically is open.

(TP5) Binding schedulability to safety across the MRM mode change: What does a timing violation *mean* for safety, and what must a timing analysis guarantee when a miss becomes imminent? A deadline miss or stale datum acquires meaning only once tied to fallback, degraded modes, and the MRM; therefore, the analysis should cover the full fail-safe sequence: detecting an impending miss at the optimal instant (too late leaves no time for a safe reaction; too early causes spurious mode changes), triggering the mode change that starts the MRM, and guaranteeing MRM schedulability after the change. Schedulability analysis [11], runtime miss detection [24], [25], and mixed-criticality scheduling [23] each address one stage in isolation; a model that co-derives all three remains open.

V. OPEN PROBLEMS: FOR PRACTITIONERS

The second set of open problems targets *practitioners* who build and deploy AD software on ROS 2 and Autoware. Where Section IV asks what model is missing, each problem below asks how to *engineer* a real system that is at once analyzable, performant, and safe. Each is either the practitioner-side dual of a research problem or a necessary condition for applying the analyses those problems call for to a real system. As duals, heterogeneous resource modeling (TP3) meets resource isolation and middleware transparency (PP1, PP3), and binding schedulability to safety across the MRM mode change (TP5) meets online violation detection (PP4). As necessary conditions, no analysis applies unless the implementation faithfully realizes the task model (PP2) and the platform enforces the scheduler that the analysis assumes (PP5). Progress on one side is therefore useful only if matched on the other.

(PP1) Eliminating non-essential complexity: How can we remove the non-essential complexity that the middleware

layer imposes on scheduling? In ROS 2, nested scheduling (the ROS 2 executor on top of the OS thread scheduler) is an implementation artifact rather than an inherent necessity. Establishing a persistent one-to-one correspondence between callbacks and OS threads lets OS scheduling parameters apply directly per callback and lets analysis ignore the executor altogether [36]. Communication is a second source: DDS runs its own internal threads for message transport, invisible to the scheduling model yet contending for the same cores; adopting true zero-copy IPC such as Agnocast eliminates these dedicated DDS communication threads [42]. Even so, residues remain: a mutually exclusive CallbackGroup serializes its callbacks outside the view of the OS scheduler, an overrun of a non-reentrant callback reintroduces middleware-level queueing, and discovery, services, and inter-host transport still occupy DDS threads. Eliminating these residues and characterizing exactly when the middleware layer can be ignored is open.

(PP2) Enforcing the implementation–model contract: How can the implementation be shaped so that it faithfully realizes the theoretical task model? Task models presume that a task makes its outputs visible to successors only upon completion, yet the ROS 2 publish/subscribe API does not force the message sending of a task to occur at its end: a `publish` may fire at any point during execution. This unconstrained send timing breaks not only DAG precedence (completion boundaries and join synchronization) but the precedence assumptions of task models in general; hence, the model holds only by programmer discipline and collapses once violated. Expressing each subtask as a function whose arguments and return values are its incoming and outgoing edges binds message exchange to subtask completion at the API rather than by convention, opening a direct path from scheduling theory to practice [59]. Extending this to message synchronization, shared-variable, and blocking flows and deploying it on production middleware is open.

(PP3) Separating functions of different criticality: How can functions of different criticality (the safety-critical MRM at the highest level, the nominal AD functions of perception, planning, and control at an intermediate level, and logging or a human-machine interface at the lowest) share one platform without the lower-criticality ones disturbing the higher-criticality ones? The underlying tension is between *partitioning* for assurance and *sharing* for efficient resource usage [23]: full physical separation isolates criticalities but squanders the limited compute, power, and weight budget of an embedded platform; the goal is therefore to consolidate functions on shared hardware while still guaranteeing isolation. Commodity ROS 2 on Linux offers only partial mechanisms: running real-time DAGs under FIFO/EDF while confining best-effort work to the Completely Fair Scheduler (CFS) on exclusive CPU affinities [36], hosting on heterogeneous SoCs [49], and virtualized or containerized deployment [10], [50]. Enforcing temporal and spatial isolation by criticality (without statically over-provisioning away the efficiency that motivated sharing) and proving it holds on such commodity stacks is open.

(PP4) Detecting timing-constraint violations early: How can we detect, at runtime and early enough to act, that a timing constraint will be missed? Post-hoc detection is too late for a safe transition. Early-detection methods for DAG tasks with variable and probabilistic thresholds [24], [25] and runtime miss/freshness tracking in tools such as TILDE [46] point the way. A low-overhead, chain- and graph-level online detector wired to fallback and deployable in production remains open.

(PP5) Implementing state-of-the-art schedulers: How can the latest scheduling algorithms actually be enforced on commodity middleware and OS? The underlying obstacle is that no system software treats a timing-constrained unit of execution (a DAG, for instance) as a *first-class citizen*: an entity the runtime maintains natively and whose precedence and deadline semantics the scheduler enforces directly, rather than one reconstructed by convention atop primitives (threads or callbacks) that have no notion of it. Lacking such a substrate, published DAG schedulers cannot be enforced as specified and are evaluated largely by analysis or simulation. Enabling mechanisms are emerging: per-callback OS parameters [36], chain-aware priorities [13], flexible OS-level scheduling [35], and reconciliation with classical periodic scheduling [15]. None of them, however, makes the timing-constrained unit first-class. A repeatable path from a published algorithm to a low-overhead, enforced implementation on production stacks is open.

VI. Conclusion

This paper revisited real-time models for AD systems from both the theoretical and the implementation/evaluation side. On the theory side, work built on periodic/sporadic tasks, DAGs, pipelines, mixed-criticality, and timer-/event-driven activation is evolving toward E2E latency along cause-effect chains, data freshness, timing disparity, probabilistic timing, highest-criticality fail-safe operation, and early miss detection. On the implementation side, Autoware, ROS 2 executors, ROS-lite and ROS-lite2, Agnocast, and tracing/evaluation tools such as CARET, TILDE, ros2_tracing, and Autoware_Perf measure and improve real-time behavior on real systems. A practical real-time task and timing model for AD systems is therefore not a single task model but a layered framework spanning callback/graph/chain units, multi-rate/event-driven execution, chain-aware metrics, heterogeneous resources, variability-aware timing, and a connection to safety control. Closing the theory–reality gap is a two-way convergence: theoretical models must evolve toward AD reality, and Autoware/ROS 2 implementations must be reshaped to realize analyzable models. The key next step is to make both sides move by binding these models to Autoware/ROS 2 tooling while reshaping that tooling to be analyzable, so that theory can inform practical design and practice can ground theory. We framed these next steps as two parallel sets of open problems: for the theory community (the right unit of analysis, joint metrics, heterogeneous resources, execution-time variability, and the safety connection) and for practitioners (removing non-essential complexity, closing the implementation–model gap, isolating functions of

different criticality, online violation detection, and deploying state-of-the-art schedulers).

Acknowledgments

This work was supported by JST FOREST Grant Number JPMJFR242G, and is based on results obtained from a project, Green Innovation Fund Projects / Development of In-vehicle Computing and Simulation Technology for Energy Saving in Electric Vehicles (JPNP21027), subsidized by the New Energy and Industrial Technology Development Organization (NEDO).

References

[1] Open Robotics, "rclcpp: Template class subscription `take`," 2024, accessed: Jun. 18, 2026. [Online]. Available: https://docs.ros.org/en/humble/p/rclcpp/generated/classrclcpp_1_1Subscription.html.

[2] ——, "message_filters: `ExactTime` and `ApproximateTime` synchronization policies," 2024, accessed: Jun. 18, 2026. [Online]. Available: https://wiki.ros.org/message_filters.

[3] R. Kambe, M. Kurihara, T. Kawaguchi, R. Kawabuchi, and Y. Takano, "Redundant autonomous driving system with formally verified leader election algorithm," *Journal of Information Processing*, vol. 33, pp. 890–900, 2025.

[4] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[5] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993.

[6] A. Yano and T. Azumi, "Work-in-progress: Multi-deadline DAG scheduling model for autonomous driving systems," in *Proc. of the 45th IEEE Real-Time Systems Symposium (RTSS), Work-in-Progress Session*, 2024, pp. 451–454.

[7] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, vol. 80, pp. 104–113, 2017.

[8] H. Teper, M. Günzel, N. Ueter, G. von der Brüggen, and J.-J. Chen, "End-to-end timing analysis in ROS2," in *Proc. of the 43rd IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 53–65.

[9] M. Günzel and M. Becker, "Optimal task phasing for end-to-end latency in harmonic and semi-harmonic automotive systems," in *Proc. of the 31st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2025, pp. 164–176.

[10] T. Betz, H. Teper, D. Ebner, M. Leitenstern, S. Sagmeister, M. Weinmann, J.-J. Chen, and M. Lienkamp, "End-to-end latency optimization for containerized ROS 2 autonomous driving software," *IEEE Access*, vol. 13, pp. 112 654–112 672, 2025.

[11] J. Sun, K. Duan, X. Li, N. Guan, Z. Guo, Q. Deng, and G. Tan, "Real-time scheduling of autonomous driving system with guaranteed timing correctness," in *Proc. of the 29th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2023, pp. 185–197.

[12] T. Kobayashi, R. Okamura, and T. Azumi, "Response time analysis with cause-effect chain considering DAG structure and high-load tasks," *IEEE Access*, vol. 13, pp. 21 557–21 568, 2025.

[13] H. Choi, Y. Xiang, and H. Kim, "PiCAS: New design of priority-driven chain-aware scheduling for ROS2," in *Proc. of the 27th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 251–263.

[14] H. Sobhani, H. Choi, and H. Kim, "Timing analysis and priority-driven enhancements of ROS 2 multi-threaded executors," in *Proc. of the 29th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2023, pp. 106–118.

[15] H. Teper, O. Bell, M. Günzel, C. Gill, and J.-J. Chen, "Reconciling ROS 2 with classical real-time scheduling of periodic tasks," in *Proc. of the 31st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2025, pp. 177–189.

[16] R. Li, N. Guan, X. Jiang, Z. Guo, Z. Dong, and M. Lv, "Worst-case time disparity analysis of message synchronization in ROS," in *Proc. of the 43rd IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 40–51.

[17] Y. Tang, X. Jiang, N. Guan, X. Luo, M. Yang, and W. Yi, "Timing analysis of processing chains with data refreshing in ROS 2," *Journal of Systems Architecture*, vol. 155, p. 103259, 2024.

[18] J. Sun, X. Li, M. Gong, N. Guan, Z. Guo, M. Chen, J. Zhao, and Q. Deng, "Jointly ensuring timing disparity and end-to-end latency constraints in hybrid DAGs," in *Proc. of the 31st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2025, pp. 190–201.

[19] M. Günzel, N. Ueter, K.-H. Chen, G. von der Brüggen, and J.-J. Chen, "Probabilistic reaction time analysis," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 5s, pp. 143:1–143:22, 2023.

[20] T. Han and K. Kim, "Minimizing probabilistic end-to-end latencies of autonomous driving systems," in *Proc. of the 29th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2023, pp. 27–39.

[21] S. J. Gil, I. Bate, G. Lima, L. Santinelli, A. Gogonel, and L. Cucu-Grosjean, "Open challenges for probabilistic measurement-based worst-case execution time," *IEEE Embedded Systems Letters*, vol. 9, no. 3, pp. 69–72, 2017.

[22] S. Bozhko, F. Marković, G. von der Brüggen, and B. B. Brandenburg, "What really is pWCET? A rigorous axiomatic proposal," in *Proc. of the 44th IEEE Real-Time Systems Symposium (RTSS)*, 2023, pp. 13–26.

[23] A. Burns and R. I. Davis, "Mixed criticality systems — a review," Department of Computer Science, University of York, Tech. Rep., 2022, 13th edition.

[24] H. Toba and T. Azumi, "Deadline miss early detection method for DAG tasks considering variable execution time," in *Proc. of the 36th Euromicro Conference on Real-Time Systems (ECRTS)*, ser. LIPIcs, vol. 298, 2024, pp. 8:1–8:21.

[25] H. Toba, S. Baruah, and T. Azumi, "Enhancing deadline miss early detection for DAG tasks with variable probabilistic thresholds," *ACM Transactions on Embedded Computing Systems*, 2026, just accepted.

[26] M. Alcon, E. Mezzetti, J. Abella, and F. J. Cazorla, "Supporting timing-related metrics for autonomous driving frameworks in CyberRT," *ACM Transactions on Design Automation of Electronic Systems*, vol. 31, no. 1, pp. 6:1–6:37, 2026.

[27] T. Azumi, Y. Maruyama, and S. Kato, "ROS-lite: ROS framework for NoC-based embedded many-core platform," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 4375–4382.

[28] F. Reghenzani, L. Santinelli, and W. Fornaciari, "Dealing with uncertainty in pWCET estimations," *ACM Transactions on Embedded Computing Systems*, vol. 19, no. 5, pp. 33:1–33:23, 2020.

[29] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement," in *Proc. of the 26th Euromicro Conference on Real-Time Systems (ECRTS)*, 2014, pp. 109–118.

[30] G. Brilli, R. Cavicchioli, M. Solieri, P. Valente, and A. Marongiu, "Evaluating controlled memory request injection for efficient bandwidth utilization and predictable execution in heterogeneous SoCs," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 1, pp. 7:1–7:25, 2023.

[31] S. Bateni, Z. Wang, Y. Zhu, Y. Hu, and C. Liu, "Co-optimizing performance and memory footprint via integrated CPU/GPU memory management, an implementation on autonomous driving platform," in *Proc. of the 26th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 310–323.

[32] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *Proc. of the 9th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 287–296.

[33] Y. Yang and T. Azumi, "Exploring real-time executor on ROS 2," in *Proc. of the 16th IEEE International Conference on Embedded Software and Systems (ICESS)*, 2020, pp. 1–8.

[34] B. Peng, Y. Yang, Y. Okumura, A. Hasegawa, and T. Azumi, "ThreadedCallback: Improving real-time performance of ROS 2," in *Proc. of the Asia Pacific Conference on Robot IoT System Development and Platform (APRIS)*, 2021, pp. 34–41, IPSJ. Accessed: Jun. 12, 2026. [Online]. Available: https://ipsj.ixsq.nii.ac.jp/records/216182.

[35] S. Liu, R. Wagle, S. Ahmed, Z. Tong, and J. H. Anderson, "ROSRT: Enabling flexible scheduling in ROS 2," in *Proc. of the 46th IEEE Real-Time Systems Symposium (RTSS)*, 2025, pp. 42–54.

[36] T. Ishikawa-Aso, A. Yano, T. Azumi, and S. Kato, "Work in progress: Middleware-transparent callback enforcement in commoditized

component-oriented real-time systems," in *Proc. of the 31st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2025, pp. 426–429.

[37] Y. Saito, F. Sato, T. Azumi, S. Kato, and N. Nishio, "ROSCH: Real-time scheduling framework for ROS," in *Proc. of the 24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018, pp. 52–58.

[38] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," in *Proc. of the 13th ACM International Conference on Embedded Software (EMSOFT)*, 2016, pp. 1–10.

[39] Y. Tajima, S. Tsunoda, and T. Azumi, "ROS-lite2: Autonomous-driving software platform for clustered many-core processor," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 4578–4585.

[40] Y.-P. Wang, W. Tan, X.-Q. Hu, D. Manocha, and S.-M. Hu, "TZC: Efficient inter-process communication for robotics middleware with partial serialization," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 7805–7812.

[41] M. Pöhnl, C. Eltzschig, and T. Blass, "Shared-memory-based lock-free queues: The key to fast and robust communication on safety-critical edge devices," in *Proc. of the ACM Cyber-Physical Systems and Internet of Things Week (CPS-IoT Week)*, 2023, pp. 179–184.

[42] T. Ishikawa-Aso and S. Kato, "ROS 2 Agnocast: Supporting unsized message types for true zero-copy publish/subscribe IPC," in *Proc. of the 28th IEEE International Symposium on Real-Time Distributed Computing (ISORC)*, 2025, pp. 1–10.

[43] S. Kim, J. Song, K. Lee, S. Oh, and H. S. Chwa, "CROS-RT: Cross-layer priority scheduling for predictable inter-process communication in ROS 2," in *Proc. of the 31st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2025, pp. 230–242.

[44] C. Bédard, I. Lütkebohle, and M. Dagenais, "ros2_tracing: Multipurpose low-overhead framework for real-time tracing of ROS 2," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6511–6518, 2022.

[45] T. Kuboichi, A. Hasegawa, B. Peng, K. Miura, K. Funaoka, S. Kato, and T. Azumi, "CARET: Chain-aware ROS 2 evaluation tool," in *Proc. of the 20th IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, 2022, pp. 1–8.

[46] X. He, H. Sato, Y. Okumura, and T. Azumi, "TILDE: Topic-tracking infrastructure for dynamic message latency and deadline evaluator for ROS 2 application," in *Proc. of the 27th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2023, pp. 1–9.

[47] Z. Li, A. Hasegawa, and T. Azumi, "Autoware_Perf: A tracing and performance analysis framework for ROS 2 applications," *Journal of Systems Architecture*, vol. 123, p. 102341, 2022.

[48] A. Yano and T. Azumi, "RD-Gen: Random DAG generator considering multi-rate applications for reproducible scheduling evaluation," in *Proc. of the 26th IEEE International Symposium on Real-Time Distributed Computing (ISORC)*, 2023, pp. 21–31.

[49] H. Chishiro, K. Suito, T. Ito, S. Maeda, T. Azumi, K. Funaoka, and S. Kato, "Towards heterogeneous computing platforms for autonomous driving," in *Proc. of the 15th IEEE International Conference on Embedded Software and Systems (ICESS)*, 2019, pp. 1–8.

[50] L. Wen, M. Rickert, F. Pan, J. Lin, and A. Knoll, "Bare-metal vs. hypervisors and containers: Performance evaluation of virtualization technologies for software-defined vehicles," in *Proc. of the IEEE Intelligent Vehicles Symposium (IV)*, 2023, pp. 1–8.

[51] P. H. E. Becker, J.-M. Arnau, and A. González, "Demystifying power and performance bottlenecks in autonomous driving systems," in *Proc. of the IEEE International Symposium on Workload Characterization (IISWC)*, 2020, pp. 205–215.

[52] J. Peeck, J. Schlatow, and R. Ernst, "Online latency monitoring of time-sensitive event chains in safety-critical applications," in *Proc. of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 539–542.

[53] T. Betz, M. Schmeller, H. Teper, and J. Betz, "How fast is my software? Latency evaluation for a ROS 2 autonomous driving software," in *Proc. of the IEEE Intelligent Vehicles Symposium (IV)*, 2023, pp. 1–6.

[54] H. Teper, T. Betz, G. von der Brüggen, K.-H. Chen, J. Betz, and J.-J. Chen, "Timing-aware ROS 2 architecture and system optimization," in *Proc. of the 29th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2023, pp. 206–215.

[55] H. Abaza, D. Roy, S. Fan, S. Saidi, and A. Motakis, "Trace-enabled timing model synthesis for ROS2-based autonomous applications," in *Proc. of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.

[56] G. Sciangula, D. Casini, A. Biondi, and C. Scordino, "End-to-end latency optimization of thread chains under the DDS publish/subscribe middleware," in *Proc. of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.

[57] C. Fan, L. Nie, J. Zhang, K. Dai, S. Gao, and J. Li, "Uncovering underexplored runtime behaviors in ROS2-based autonomous systems," *ACM Transactions on Internet of Things*, vol. 7, no. 3, 2026.

[58] C. Bédard, P.-Y. Lajoie, G. Beltrame, and M. Dagenais, "Message flow analysis with complex causal links for distributed ROS 2 systems," *Robotics and Autonomous Systems*, vol. 161, p. 104361, 2023.

[59] T. Ishikawa-Aso, A. Yano, Y. Kobayashi, T. Jin, Y. Takano, and S. Kato, "Work-in-progress: Function-as-subtask API replacing publish/subscribe for OS-native DAG scheduling," in *Proc. of the 46th IEEE Real-Time Systems Symposium (RTSS), Work-in-Progress Session*, 2025, pp. 604–607.