AD-TTS: A Multi-Task Simulation Framework and Dataset for Event Prediction in Time-Triggered Autonomous Driving

Dongchen Li
Chair for Embedded Systems
University of Siegen
Siegen, Germany
Dongchen.Li@uni-siegen.de
ORCID: 0009-0001-1011-4285

Daniel Onwuchekwa
Chair for Embedded Systems
University of Siegen
Siegen, Germany
Daniel.Onwuchekwa@uni-siegen.de
ORCID: 0009-0003-7220-8265

Roman Obermaisser
Chair for Embedded Systems
University of Siegen
Siegen, Germany
roman.obermaisser@uni-siegen.de
ORCID: 0009-0002-4483-1503

***Abstract*—Time-triggered systems are widely deployed in safety-critical applications due to their deterministic timing behavior. To enhance adaptability under dynamic workloads and environmental changes, adaptive time-triggered systems have received increasing attention. Existing approaches either consume excessive memory or introduce adaptation delays. Proactive speculative scheduling, driven by event prediction, has emerged as a promising alternative. Its core component is an event prediction model. Despite the availability of various time-series and event log datasets in other domains, no suitable dataset exists for event prediction in time-triggered systems. This paper introduces Autonomous Driving Time-Triggered System Dataset (AD-TTS), a dataset tailored for learning-based event prediction in multi-task time-triggered systems. We extend an autonomous driving simulation from CARLA into a multi-task time-triggered system that consists of 21 tasks and 32 event types. Events are logged at each scheduling cycle along with metadata such as timestamps, task identifiers, CPU utilization, memory usage, and event attributes. AD-TTS provides a foundation for future research on event prediction models and adaptive scheduling techniques in time-triggered systems.**

***Index Terms*—dataset construction, time-triggered systems, autonomous driving simulation, multi-task scheduling, event logging**

I. Introduction

As time-triggered system scenarios grow more complex, task counts, resource demands, and failure rates are rising rapidly. Adaptive time-triggered systems address these challenges by switching schedules based on environmental and system changes. There are two main approaches. Multi-Schedule-Graph-based (MSG) scheduling methods [1], [2] precompute and store all schedules offline, and select one at runtime based on events. However, this method consumes significant memory. In contrast, online schedule generation methods [3] periodically invoke a dedicated task to generate new schedules, offering high flexibility but also introducing adaptation delays.

Proactive speculative scheduling, an emerging method based on event prediction, addresses the aforementioned issues. The event prediction module forecasts the most likely event sequences in the upcoming cycle. Based on this prediction, two design approaches can be adopted. The first approach, similar to MSG-based approaches, precomputes all possible schedules offline and stores them on the secondary memory. At runtime, according to the prediction, the corresponding schedules are pre-loaded into the faster main memory, enabling quick switching when needed. The second approach generates new schedules proactively at runtime according to the prediction, which can reduce adaptation delays. Thus, effective event prediction is essential for enabling proactive speculative scheduling in time-triggered system.

Public datasets rarely include the task-level information needed for event prediction in time-triggered scheduling research. To fill this gap, we extended an autonomous driving (AD) simulation application provided by the CARLA simulator [4] into a time-triggered system (TTS) and generated our dataset, AD-TTS, which is publicly available at: https://github.com/EmbeddedSystems-UniversitySiegen/ADTTS-Time-Triggered-System-Dataset.

AD-TTS includes 32 distinct event types from 21 tasks, each represented as a structured event vector. Some frequent events, such as "event_task_start", occur every cycle and offer limited value for adaptive scheduling. In contrast, certain unique events driven by environmental input can influence task execution and potentially trigger schedule switching. Data analysis reveals strong inter-event correlations and periodic patterns, highlighting the feasibility of accurate event prediction.

Our main contributions are summarized as follows:

1) We extended an AD simulation into a multi-task time-triggered system based on the CARLA simulator [4] and ROS [5]. Since building a complete AD application is highly complex and beyond the scope of this work, we leveraged simulation data to represent the task execution outcomes. Each task was assigned a computational load based on its input, enabling realistic simulation of execution behavior.

2) We develop and release AD-TTS, the public dataset capturing task-level events in a time-triggered system, partitioned into five subsets to facilitate future research on event prediction models.

The remainder of this paper is organized as follows: Section II reviews existing datasets and highlights the gap motivating our work. Section III introduces the AD simulation application used as the basis for our framework. Section IV describes the design of the multi-task time-triggered system and event collection. Section V focuses on dataset preprocessing and the construction of five sub-datasets, and provides example figures to illustrate their key characteristics. Section VI concludes the paper.

II. Related Work

Existing datasets are usually designed for specific tasks. For example, datasets used to study attacks in the Internet of Things (IoT), such as IoT-SH [6], Kitsune [7], and MedBIoT [8], mainly contain non-periodic events. The work in [9] generated a real-time scheduling dataset for 5G, including features such as network resource usage and packet transmission counts. Sensor datasets, such as ultrasonic sensor datasets from robots [10] or time-series sensor data from a chemical plant [11], contain large amounts of real-time environmental data.

There are also more complex publicly available datasets. The EVIoT-PredictiveMain dataset [12] contains real-world data collected from IoT-enabled electric vehicles, including 30 features, such as timestamps, battery status, and electric motor status. Each record in this dataset is sampled at 15-minute intervals. It is primarily used to train machine learning models for fault prediction and energy efficiency optimization, which helps improve vehicle reliability. A similar dataset is [13], which includes real-world charging data from an electric vehicle. It consists of 28 features such as date, battery capacity, charging time, waiting time, environmental variables (e.g., temperature and wind speed), and vehicle location. In this dataset, each record is sampled at one-hour intervals, and it is mainly used to analyze factors that influence electric vehicle charging behavior.

While these datasets are based on periodic data collection, they do not involve scheduling systems and thus cannot be directly used in our research. However, they inspired the design of our data collection procedures and data structures. These datasets can be seen as local data collected by each task within a scheduling system. At the end of each scheduling cycle, all local data are aggregated into global data. Each record, similar to those in [12], [13], includes a timestamp, event, system status, and event attributes. The data structure not only includes the periodic information of individual tasks, but also reflects inter-task relationships, thereby revealing the dynamic state transitions of the scheduling system.

Our review is limited to publicly available datasets rather than deployed time-triggered systems. Although many safety-critical applications (e.g., in automotive or aerospace domains) rely on time-triggered scheduling, their event logs are rarely released with the task-level parameters, dependencies, and execution states required for event prediction research. Available logs typically contain only anonymized event identifiers and timestamps, which are insufficient to reconstruct the scheduling context or evaluate predictive methods. This lack of task-level, system-wide datasets constitutes a critical research gap and directly motivates the development of the AD-TTS simulation framework and dataset.

III. Autonomous Driving Simulation Platform

To generate datasets for event prediction, we extended an AD simulation application based on the CARLA simulator [4] and ROS [5]. The CARLA simulator is an open-source platform specifically designed for AD research and provides several reference simulation applications. One of these was adapted into a time-triggered system comprising 21 tasks that communicate via ROS. These tasks were distributed across 16 CPU cores on a dedicated server with 32 GB of RAM, an Intel Core i7-12700K (20-core CPU), and an NVIDIA GeForce RTX 3070. The operating system is 64-bit Ubuntu 20.04.6 LTS with the 5.15.133-rt69 real-time Linux kernel.

Each AD simulation utilizes different configurations, including random numbers of vehicles and pedestrians, a pair of randomly selected departure and destination points for the ego vehicle (the vehicle being controlled and observed in the simulation), as well as random maps and weather conditions. Therefore, each dataset generated by the simulation is unique.

A. Sensor Configuration

The sensor configuration used on the simulated autonomous vehicle is inspired by that of the Tesla Model 3 [1], as shown in Fig. 1. Some sensors may have sampling periods longer than a single scheduling cycle. To avoid excessive simulation time and data volume, the sampling periods of the sensors are not set to match the typical values of their real-world counterparts.

The sampling period is defined in "task seconds", which corresponds to the time-triggered system. Unless otherwise specified, the term "second" refers to "task seconds" throughout this paper. CARLA's rendering time varies depending on the number of 3D objects in the scene and is unrelated to the scheduling system. Therefore, we adopt two separate time systems to manage the simulation environment and the time-triggered system, respectively, as discussed in Section IV-B.

Each simulated sensor is assigned a small probability of data loss to simulate signal interference or data conflicts. In real-world scenarios, the likelihood of data loss can vary significantly depending on sensor types and models. However, this paper specifically focuses on the impact of sensor data loss on subsequent triggered events. Therefore, the probability of data loss is configurable in our framework, and the default value is set to 0.1% in this study.

[1]We referred to a teardown blog describing the sensor and chip components used in Tesla Model 3 [14]. Although the source is non-academic and in Chinese, it provided useful insights for designing a plausible sensor configuration for our simulation prototype. We emphasize that this reference was used solely for implementation guidance and does not influence the theoretical or empirical results of this study.

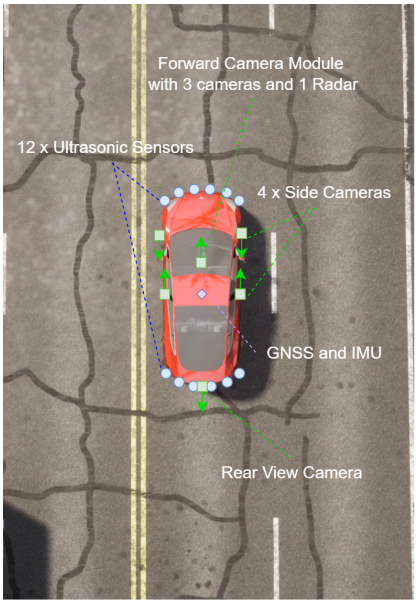

Fig. 1. Diagram of sensor installation locations. There are six cameras, 12 ultrasonic sensors, one radar, one GNSS, and one IMU.

The navigation module includes a Global Navigation Satellite System (GNSS) sensor and an Inertial Measurement Unit (IMU) sensor. The GNSS provides vehicle positioning information every 5 seconds [2], with a data loss probability of 0.3%. This relatively high loss rate reflects the greater likelihood of signal interference in satellite communication compared to direct hardware connections. The IMU sensor collects inertial data, including three-axis acceleration, three-axis angular velocity, and magnetic field orientation, and provides data at every scheduling cycle.

A radar sensor is installed on the front of the vehicle with a detection range of 160 meters. The radar model provided by CARLA simulates the actual operation of real-world radar systems. It scans a 35-degree horizontal field of view and a 5-degree vertical field of view, generating a point cloud as output data. Additionally, six ultrasonic sensors are separately mounted on both the front and rear bumpers of the vehicle. These 12 sensors have a detection range of 8 meters and are used to detect obstacles in front of and behind the vehicle. Since CARLA does not include a built-in ultrasonic sensor model, the radar model is used as a substitute. Both the radar and ultrasonic sensors operate with a sampling period of one scheduling cycle. However, as their outputs are unrelated to event prediction, they are not recorded in the final event logs.

Three cameras are installed on the vehicle's front windshield: a wide-angle forward camera with a 120-degree field of view (FOV), a main forward camera with a 60-degree FOV, and a narrow forward camera with a 35-degree FOV. On each side of the vehicle, two 60-degree FOV cameras are installed to monitor the front and rear lateral areas. Additionally, a rear-facing camera with a 120-degree FOV is installed above the rear license plate to cover the vehicle's rear blind spot. Image data from these eight cameras is sampled every 10 seconds. Since they are not relevant to event prediction, they are excluded from the final event logs.

[2]The sampling periods of both GNSS and camera sensors far exceed those of real sensors, as executing all tasks within a scheduling cycle on our server takes approximately 830 ms (task clock). Therefore, we used CARLA's synchronous mode to pause the simulation while the scheduling system runs. As a result, the sampling periods of the GNSS and camera sensors are about 140 and 260 ms (simulation clock), respectively. This approach ensures sufficient time to complete all tasks without adversely affecting the control system.

B. Autonomous Driving Logic

CARLA's "autopilot" example was extended to develop our AD simulation application. CARLA provides rich simulation environment information, including the states of each traffic light, road signs, waypoints [3], and the states of surrounding vehicles. The path planning module connects the start and destination using a sequence of waypoints and continuously updates the ego vehicle's intended direction. When nearby vehicles or obstacles are detected, the ego vehicle performs evasive maneuvers. The low-level controller is a PID controller. Given the planned direction, vehicle state, and surroundings, it calculates the deviation and determines the next waypoint. Throttle, steering, and braking signals are adjusted accordingly to navigate around obstacles.

Basic traffic rules are enforced. The system identifies the next relevant traffic light by comparing the vehicle's direction with the positions of traffic lights. When approaching an intersection, the ego vehicle determines whether to stop or proceed according to standard rules—stopping at red lights and proceeding at green. Using a similar method, the system can detect pedestrians ahead and initiate evasive actions when necessary. The road sign information provided by the CARLA simulator indicates whether adjacent lanes are available. A dashed line denotes an available lane, while a solid line indicates none. If the ego vehicle has been following another vehicle for a long period and an adjacent lane is available, an overtaking maneuver is initiated. The control strategy for overtaking is similar to that used for obstacle avoidance.

This simulation does not implement a full AD system. Instead, it generates control signals based on information directly provided by the simulation environment to guide the vehicle to its destination. Although individual modules can be realized using existing solutions—for example, vehicle detection tasks are often implemented using YOLO models—the development and integration of a full AD pipeline remain significant challenges and are beyond the scope of this study.

IV. Scheduling System and Event Logging

A. Task Set

Our AD simulation application consists of 21 tasks, which communicate with each other via ROS. The task dependency graph is shown in Fig. 2. By referring to the configuration

[3]In CARLA's built-in maps, each lane is associated with a sequence of points called waypoints. Each waypoint includes information about position and lane direction, which helps the control program quickly identify lanes and determine their directions.

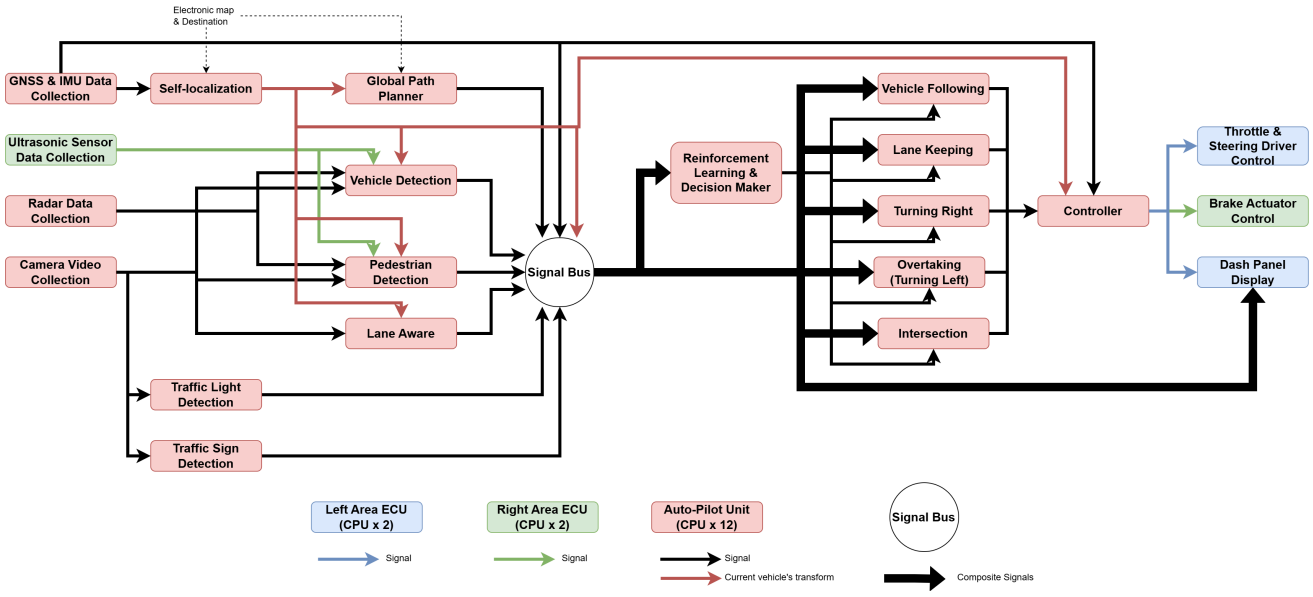

Fig. 2. Message dependencies among 21 tasks in the AD simulation application. These tasks are assigned to two Electronic Control Units (ECUs) and an Auto-Pilot Unit (AP), each with a different number of CPU cores.

of the Tesla Model 3, we simulated three computing modules and assigned them different numbers of CPU cores: the Left Area ECU (2 cores), the Right Area ECU (2 cores), and the Auto-Pilot Unit (12 cores). Tasks assigned to each module are restricted to execute exclusively on the cores allocated to their respective module.

Since commercial AD software and detailed design documents are not publicly available, the logical implementation of all tasks—except for sensor collection tasks, the "Global Path Planner Task", and the "Controller Task"—is based on the simulation environment data rather than complete code. Although the functionalities are simplified, the core data characteristics are preserved. For example, in the "Vehicle Detection Task", the positions of all vehicles in the environment are obtained directly via CARLA's APIs. Their distances to the ego vehicle are then computed, and any vehicle within a predefined detection range is considered detected by this task.

Since the slack time of a task is also a crucial factor affecting scheduling, computational loads are designed to simulate the actual execution behavior of tasks. A computational load is implemented as a loop that consumes processing time without performing any functional operation, and its intensity varies depending on environmental inputs and system states. Three types of computational loads are considered in this work:

1) **Periodic Load**: Sensors often collect data at a fixed frequency; thus, the computational load of sensor data collection tasks, such as "Camera Video Task", also changes periodically.
2) **Random Load**: Some sensors, such as GNSS, may lose data. The computational load of the corresponding tasks increases to simulate such behavior.
3) **Dependency Load**: The computational load of a task varies depending on its function, the received messages, and the current system state. Since these factors influence both the task's behavior and its execution time, the computational load dynamically adapts to reflect such dependencies.

The worst-case execution time (WCET) of each task is set as its longest execution time plus a 20% margin. The schedule is generated offline using a simple list scheduler and remains fixed during runtime.

B. Time Synchronization

The rendering time of the CARLA simulator is not relevant for the event prediction, while the tasks are implemented in ROS nodes that operate on a different time system. To preserve the properties of time-triggered systems and distinguish between simulation time and task execution time, three clocks are defined in the AD simulation application:

1) **Simulation clock**: The time that passes in the simulation world, retrievable via a CARLA API.
2) **Task clock**: It counts only during the task running phase. The schedule's time base is derived from it, making it unaffected by the CARLA simulator's variable rendering time.
3) **Computer clock**: A global clock that starts counting from the beginning of the application.

The relationship among the three clocks is illustrated in Fig 3. In synchronous mode, the CARLA simulator renders a fixed duration of simulation time per invocation, referred to as a tick rendering, while the actual rendering time in the real world remains variable. Each tick rendering advances the simulation clock by 20 milliseconds. During this rendering phase, the task clock remains paused. When the task clock starts advancing, the simulation clock remains static. After each rendering, both clocks remain unchanged for a short

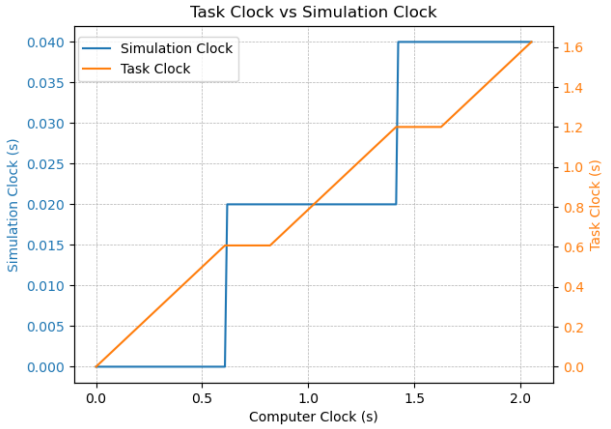

Fig. 3. Relationship among the simulation clock, task clock, and computer clock.

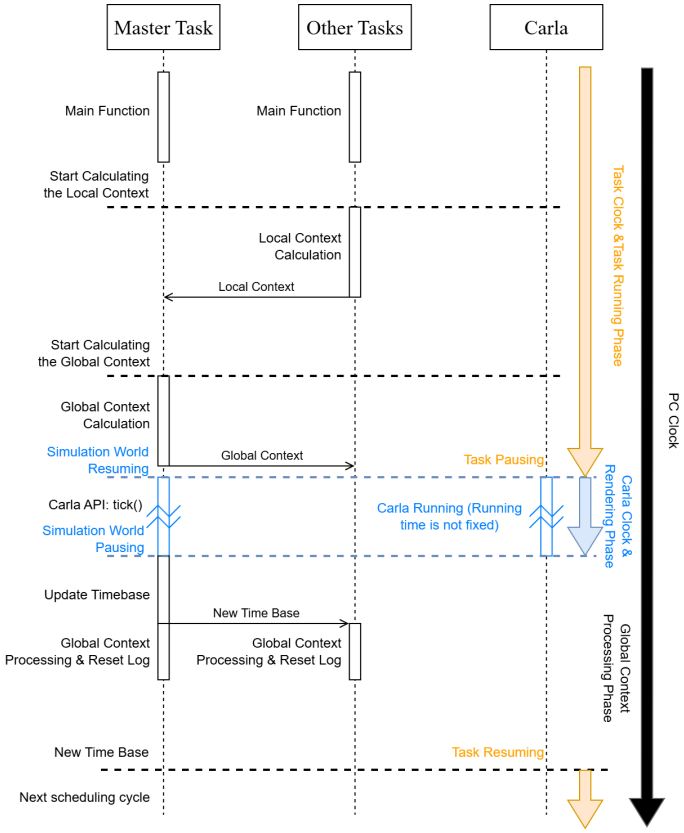

Fig. 4. Workflow of one scheduling cycle with three phases. The counting periods of the simulation clock, task clock, and computer clock (PC clock) are represented by the orange, blue, and black arrows, respectively.

period during which the collected events are processed and written to disk in real time. As shown in Fig 3, the simulation clock and task clock advance in an alternating manner.

The working flow diagram is shown in Fig 4, illustrating the operation of the three clocks throughout the scheduling cycle. A scheduling cycle is divided into three phases: the task running phase, the CARLA rendering phase, and the global context processing phase. The task clock operates independently of the simulation clock, allowing the scheduler to maintain the timing determinism characteristic of time-triggered systems.

C. Context Modeling

Each task contains an event monitoring module responsible for logging events. An entry in the event log is an *event vector*, hereafter referred to as V_e, which is formally defined in (1). It includes a one-hot encoding of the event ID (E) and the corresponding task ID (T), the associated scheduling cycle ID (C), three timestamps (computer clock t_{pc}, simulation clock t_{sim}, and task clock t_{task}), memory usage (U_{memory}), CPU utilization (U_{cpu}), CPU core ID (I_{core}), and event attribute list (A). All attribute positions in the list, $< A_1, A_2, \ldots, A_m >$, are predetermined offline and fixed at runtime. While each attribute in A corresponds to a different event type, a V_e includes all of them. Only the attributes associated with the current event ID are non-zero; the rest are set to a default value of 0.

$$V_e = (C, t_{pc}, t_{sim}, t_{task}, U_{cpu}, U_{memory}, I_{core}, T, E, A) \quad (1)$$

A sequence of event vectors is referred to as a *trace*. A trace consisting of event vectors from the same task is defined as the *local context*. As illustrated in Fig 4, at the end of each task running phase, all tasks except the master task transmit their local context to the master task [4]. These local contexts are then concatenated to form a trace that contains event vectors from all tasks, which is referred to as the *global context*. The global context generated within a single scheduling cycle is referred to as a *cycle*. It is sent back to all tasks and serves as an input for research on adaptive scheduling algorithms.

To reduce memory usage and prevent data loss caused by unexpected simulation interruptions, the global context is written to disk as an event log during the global context processing phase. Afterward, the online event log is reset. Since both the task clock and simulation clock remain frozen during this phase, the non-deterministic duration of data storage does not violate the temporal property of the time-triggered system.

D. Additional CPU Utilization Data

Although each event vector contains the CPU utilization at the moment the event occurs, the list of CPU utilizations during a task's execution period provides a more representative view of its runtime behavior. To obtain higher-frequency CPU utilization data, we developed a customized Linux kernel module to monitor all 16 CPU cores. This module is activated

[4]The master task is the "Controller Task". It has several additional responsibilities, including local context collection, global context generation, simulation world resuming, and new timebase updating. Apart from these additional responsibilities, it functions identically to other normal tasks.

at the beginning of the simulation and samples at a frequency of 1 kHz. The recorded CPU utilization data are saved as an additional system log at the end of the simulation.

V. Dataset Generation

The event log constitutes a high-dimensional and large-scale dataset. CPU utilization data in the system log is not directly usable and requires filtering for each task in every cycle to form the corresponding CPU utilization sequence. Therefore, the event log and system log are first preprocessed and combined to form a unified raw dataset. Based on the relevant data features, the raw dataset is then partitioned into five sub-datasets, allowing different models to utilize suitable input features.

A. Data Preprocessing

Records in the event log are initially unordered. Therefore, the records are first sorted by scheduling cycle ID and then by task timestamp. As the data in the first and last scheduling cycles is often incomplete, these cycles, along with other anomalous entries, are removed.

Due to the sporadic nature of event occurrences, the length of each cycle varies, necessitating padding to enable the application of deep learning models and other algorithms. To maintain consistency between the one-hot encoding of events and the event IDs used in the simulation, we extend the one-hot encoding by appending two additional dimensions for the special tokens: Start of Sequence (SOS) and End of Sequence (EOS). These tokens are then inserted at the beginning and end of each cycle to mark the valid sequence boundaries. Finally, zero vectors are padded to each cycle to match the length of the longest valid cycle, resulting in fixed-length sequences.

Most events and their corresponding attributes appear only once within a scheduling cycle. Only a few attributes are repeated, such as ”event_task_start” with the ”start_time” attribute and ”event_task_end” with the ”execution_time” attribute, which are triggered at the start and end of each task execution. These attribute values form a sparse matrix, leading to significant memory consumption. To address this issue, all attributes are extracted into a separate list to represent the global attributes of a scheduling cycle, rather than being stored within individual event vectors as local attributes. For attributes that appear multiple times, we differentiate them by prefixing the task name to ensure uniqueness. As a result, each scheduling cycle maintains a single list with 122 unique attributes, significantly reducing memory usage.

The complete CPU utilization records collected during the simulation must be decomposed and mapped to individual tasks. This decomposition process is illustrated in Fig 5. In multi-core scheduling, the CPU core assigned to a task may vary across different scheduling cycles. To begin, the time boundaries of each scheduling cycle are identified using timestamps from the event log. The utilization records of all 16 CPU cores, obtained from the system log, are then segmented according to these cycle boundaries. For each task, its start and end times, assigned core ID, and associated cycle ID are used to extract the corresponding CPU utilization segment. Since task execution durations vary, zero-padding is applied to standardize the length of all per-task, per-cycle CPU utilization sequences.

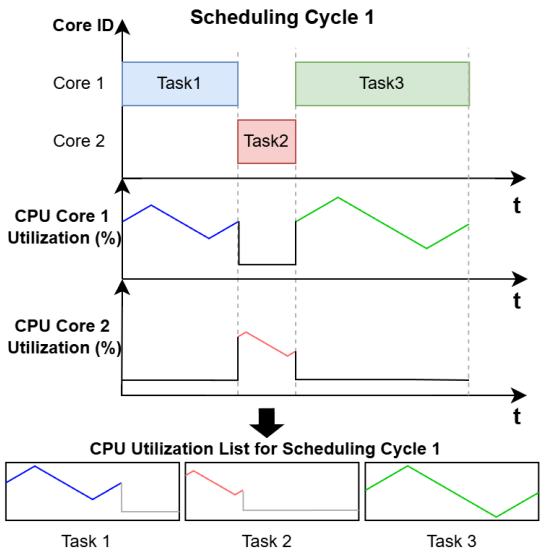

Fig. 5. Schematic diagram of CPU utilization decomposition. Three tasks run on two CPU cores. Their corresponding CPU Utilization are abstracted based on the start and end times, and then padded.

B. Sub-dataset Construction

Since the raw dataset is high-dimensional, we partitioned it into five sub-datasets based on its data characteristics to facilitate usage. The data shape of all subsets is 3-dimensional, represented as (scheduling cycle, sequence, data feature).[5]

Event sub-dataset contains only the one-hot encodings of events from the raw dataset. The shape is (6903, 82, 34), where the feature dimension of 34 consists of 32 events along with SOS and EOS tokens. Among these 32 events, some appear in every scheduling cycle, such as ”event_task_start” and ”event_task_end”, which are triggered at task dispatch and completion, respectively. These are referred to as *duplicate events*. Other events, such as ”event_vehicle_detection” and ”event_red_traffic_light”, are only triggered under certain environmental conditions and are referred to as *unique events*. These unique events often affect task execution time and may trigger subsequent related events. In adaptive time-triggered systems, they may also cause schedule switching. For instance, the occurrence of ”event_camera” indicates that new image data has been captured, which activates image recognition models in subsequent tasks like the ”Vehicle Detection Task” (for recognizing surrounding vehicles) and the ”Traffic Light Detection Task” (for detecting traffic lights), significantly in-

[5]Our GitHub repository includes multiple datasets generated from different simulation runs, each varying in total length. All dataset-related parameters reported in this paper are based on the dataset named ”2024.07.15 15h29m36s”, which contains 6905 scheduling cycles in total. After preprocessing, 6903 cycles remain. The maximum cycle length is 82.

creasing execution time. Compared to duplicate events, unique events are of higher predictive value.

Task sub-dataset contains only the one-hot encoded task representations, with a shape of (6903, 82, 21). Each element in the task sequence corresponds to the task that triggered the associated event in the event sub-dataset, rather than reflecting the execution order defined in the schedule. Since a single task can generate multiple events, the task sub-dataset contains considerable redundancy compared to the event dataset.

Header sub-dataset contains the task clock timestamp, CPU core ID, and memory usage, with a shape of (6903, 82, 3). Since the scheduling system is only concerned with the task clock t_{task}, both t_{pc} and t_{sim} are omitted.

CPU sub-dataset has two versions. The "CPU Task" sub-dataset is the raw output from the preprocessed system log, with a shape of (6903, 21, 197), which differs from other sub-datasets. The second dimension represents the 21 tasks per cycle, and the third dimension corresponds to the CPU utilization sequence of each task. However, models using this sub-dataset must extract the utilization sequence corresponding to the task sub-dataset for event prediction, which presents a significant challenge. To address this, we designed the "CPU Sequence" sub-dataset with a shape of (6903, 82, 197), where the second dimension aligns with the task sequence. The CPU utilization sequences are stacked directly according to the task sequence, as illustrated in Fig 6. Since the task sequence contains duplicate elements, the "CPU Sequence" sub-dataset includes repeated CPU utilization sequences. This design avoids forcing the model to learn unnecessarily complex cross-references, thereby improving training efficiency. Unless otherwise specified, "CPU sub-dataset" refers to the "CPU Sequence" sub-dataset throughout this paper.

Attribute sub-dataset is non-sequential data with a shape of (6903, 1, 122). Each scheduling cycle contains only one attribute list, which includes key information such as the dispatching time, actual execution time, and slack time for all 21 tasks. Slack time is the difference between WCET and actual execution time, divided by the WCET, as defined in (2). It reflects the execution status of each task and can significantly affect adaptive scheduling algorithms.

$$t_{slack} = \frac{t_{WCET} - t_{exe}}{t_{WCET}} \tag{2}$$

C. Representative Data Samples

Two periodic events with period longer than one scheduling cycle occur in the AD simulation application, namely "event_camera" and "event_gnss". When new image data is collected, the "Camera Task" requires more time to process it and generates "event_camera". Fig 7a illustrates the periodicity of this event, and the slack time decreases upon its occurrence. Fig 7b shows the relationship between the slack time of the "Self Localization Task" and two sensor events, "event_gnss" and "event_imu", generated by the "GNSS and IMU Task". Slack time of the "Self Localization Task" increases when "event_gnss" is triggered. A certain probability

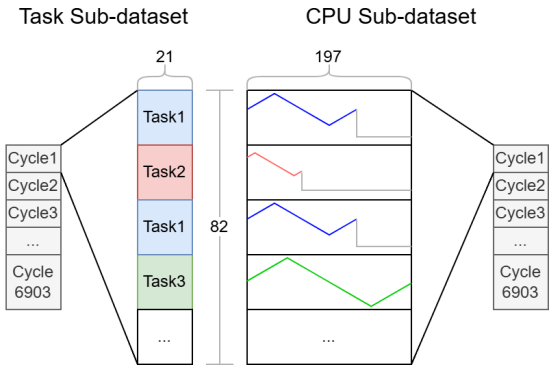

Fig. 6. Schematic diagram of stacking the CPU utilization lists based on the corresponding task sequence to generate the "CPU Sequence" sub-dataset.

of data loss was introduced in the IMU and GNSS sensors, which is reflected in the corresponding events shown in this figure.

The status of traffic lights can only be detected through image data. Therefore, "event_red_traffic_light" is triggered only when "event_camera" occurs and a red light is identified. The relationship between these two events is shown in Fig 7a. The "Decision Maker Task" generates different events based on current traffic conditions to activate corresponding functional modules. The occurrence of "event_intersection" indicates that the ego vehicle is about to enter an intersection. Consequently, the computational load of the corresponding "Intersection Task" increases significantly to handle the complex traffic situation. This relationship is shown in Fig 7d. When the ego vehicle stops at a red light, most functional modules enter a sleep state to save energy, thereby reducing the computational load of the "Decision Maker Task", as shown in Fig 7e. In contrast, when surrounding vehicles are detected, the computational load of the "Decision Maker Task" increases to maintain situational awareness, as shown in Fig 7f.

Fig 7 illustrates a strong correlation between certain unique events and task slack times, thereby suggesting the predictability of these events.

VI. CONCLUSION

To study event prediction models, we developed a simulation application based on time-triggered system and generated the AD-TTS dataset. Except for all sensor collection tasks, the "Global Path Planner Task", and the "Controller Task", other tasks were implemented using the simulation environment data provided by the CARLA APIs. However, the characteristics of task data were preserved. Computational loads were assigned to each task to simulate the task execution process. At the end of each scheduling cycle, the event vectors of all tasks were saved in real time, eventually forming the event log.

By preprocessing the event log and the CPU utilization data, the raw dataset was generated and then partitioned into five sub-datasets to facilitate subsequent research on event prediction models. We categorize events into two types: duplicate events, occurring in every scheduling cycle, and

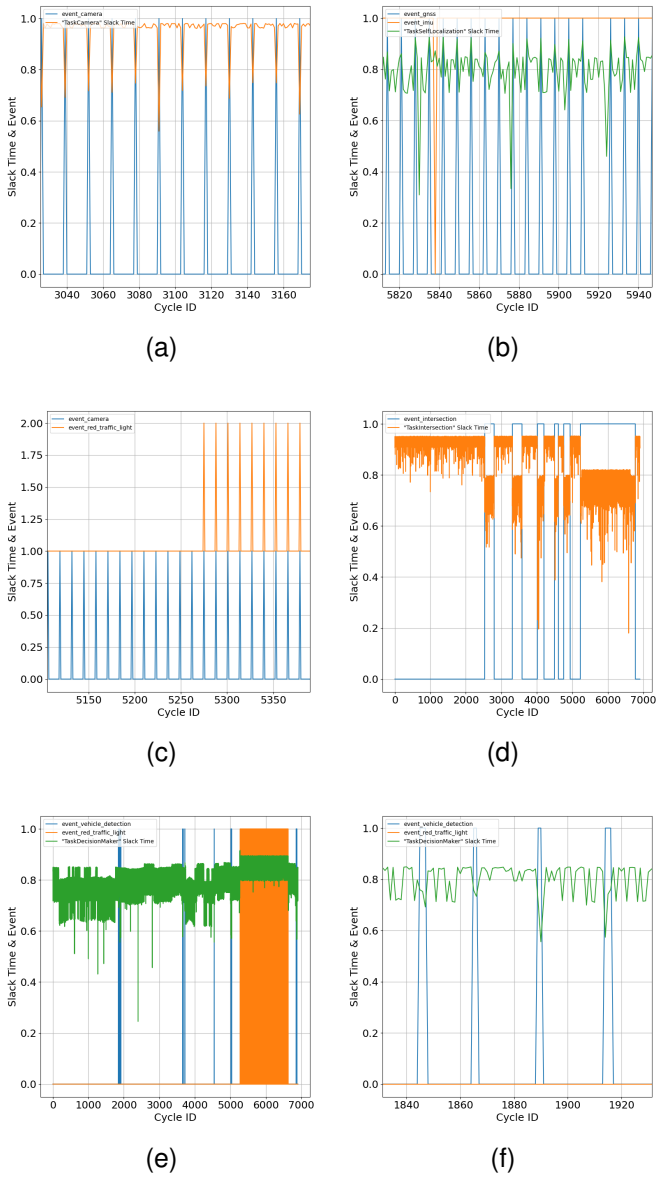

Fig. 7. Typical relationship diagram between events and task slack times. (a) shows the relationship between "event_camera" generated by the "Camera Task" and its slack time. (b) demonstrates how two events affect the slack time of "Self Localization Task". (c) illustrates that the traffic light detection relies on camera data. (d) shows that the different traffic conditions can significantly affect the task execution time. (e) illustrate a complex situation in which events from two different tasks affect the slack time from a third one. (f) is a partial enlargement of (e) from the cycle of 1800 to 2000.

unique events, triggered by environmental input changes or other events. Analysis of unique events and slack time revealed a strong correlation. Therefore, predicting unique events holds greater value for adaptive time-triggered systems. The AD-TTS dataset will serve as a solid foundation for future research on event prediction and adaptive time-triggered systems.

In future work, we plan to make our simulation architecture more representative of real-world systems by replacing the tasks currently implemented using CARLA APIs with existing solutions.

ACKNOWLEDGMENTS

This work was supported by the EdgeAI-Trust project, which is funded within the Chips Joint Undertaking (Chips JU), a public-private partnership in collaboration with the European Union's Horizon Europe research and innovation program and National Authorities under grant agreement number 101139892.

With funding from the:

Federal Ministry of Research, Technology and Space

REFERENCES

[1] P. Muoka, T. Kiesow, D. Onwuchekwa, and R. Obermaisser, *Adaptive Hierarchical Meta-Scheduling for Distributed Time-Triggered Systems with Diverse Internal and System Architectures*. Cham: Springer Nature Switzerland, 2024, pp. 117–133. [Online]. Available: https://doi.org/10.1007/978-3-031-53652-6_7

[2] P. Muoka, D. Onwuchekwa, and R. Obermaisser, "Adaptive scheduling for time-triggered network-on-chip-based multi-core architecture using genetic algorithm," *Electronics*, vol. 11, no. 1, 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/1/49

[3] K. Yang and Z. Dong, "Mixed-criticality scheduling in compositional real-time systems with multiple budget estimates," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 25–37.

[4] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[5] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[6] E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos, and P. Burnap, "A supervised intrusion detection system for smart home iot devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 9042–9053, 2019.

[7] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[8] A. Guerra-Manzanares, J. Medina-Galindo, H. Bahsi, and S. Nõmm, "Medbiot: Generation of an iot botnet dataset in a medium-sized iot network." in *ICISSP*, 2020, pp. 207–218.

[9] X. Jin, H. Chai, C. Xia, and C. Xu, "Generating datasets for real-time scheduling on 5g new radio," *Entropy*, vol. 25, no. 9, p. 1289, 2023.

[10] A. L. Freire, G. A. Barreto, M. Veloso, and A. T. Varela, "Short-term memory mechanisms in neural network learning of robot navigation tasks: A case study," in *2009 6th Latin American Robotics Symposium (LARS 2009)*, 2009, pp. 1–6.

[11] W. Hu, A. Jiang, K. Chen, J. Zheng, W. Shang, and Z. Cao, "Time-series of industrial boiler operations," 2025. [Online]. Available: https://www.kaggle.com/dsv/11704343

[12] E. D. Center, "Eviot-predictivemaint dataset," 2025. [Online]. Available: https://www.kaggle.com/dsv/10969913

[13] A. John, "Ev charging load dataset and optimal routing," 2024. [Online]. Available: https://www.kaggle.com/dsv/9604807

[14] Huizhou Camel (pseudonym), "Which chips are used in the tesla model 3? [teardown analysis]," https://aijishu.com/a/1060000000453824, 2024, aijishu Tech Blog (in Chinese), Mar. 7, 2024. Accessed: 2025-06-20.