2024 IEEE Real-Time Systems Symposium (RTSS) | 979-8-3315-4026-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/RTSS62706.2024.00033

FLEX: Adaptive Task Batch Scheduling with Elastic Fusion in Multi-Modal Multi-View Machine Perception

Yuhang Xu[†], Zixuan Liu[†], Xinzhe Fu[§], Shengzhong Liu[*†], Fan Wu[†], Guihai Chen[†]
[†]Shanghai Jiao Tong University , [§]Massachusetts Institute of Technology
Email: {xuyuhangtmx, lzx20020405}@sjtu.edu.cn, xinzhe@alum.mit.edu,
shengzhong@sjtu.edu.cn, {fwu, gchen}@cs.sjtu.edu.cn

***Abstract*—This paper presents FLEX, a real-time scheduling framework that adaptively allocates limited machine attention (*i.e.*, computing resources) among different spatial views (partitioned by camera facing directions) and sensory modalities (*i.e.*, LiDAR and cameras) within multi-modal multi-view machine perception on resource-constrained embedded platforms. It is achieved through the effective wiring of two features: First, considering the heterogeneous and time-varying criticality among views and modalities within dynamic sensing contexts (*i.e.*, object locations), we calibrate an "anytime" multi-modal perception pipeline that dynamically adjusts the modality fusion strategies of each view. Second, to optimize the GPU processing throughput with time guarantees, FLEX centers around an adaptive batch scheduling algorithm that intelligently groups consecutive asynchronous *view inspection tasks* based on the job sequence[1] generated from a non-preemptive EDF schedule to maximize a measure of system utility, with the runtime elastic fusion used as a subroutine. Temporal load balancing is maintained during the scheduling by always ensuring the sequential schedulability of future tasks in early batching decisions. We implement FLEX on NVIDIA Jetson Orin and conduct extensive experiments with a large-scale driving dataset. The results demonstrate the superiority of FLEX in improving perception quality and system throughput with time guarantees.**

***Index Terms*—Multi-Modal Perception, Task Batching, Real-Time Scheduling**

I. Introduction

Machine perception based on deep neural networks (DNNs) is widely deployed in autonomous systems such as autonomous vehicles (AV) and unmanned aerial vehicles (UAV). They are typically equipped with multiple types of sensory modalities (*e.g.*, RGB cameras and LiDAR), and rely on a 3D object detector integrating information from all modalities and different views to localize and classify surrounding objects [1]–[3]. It is imperative to achieve accurate environment perception while keeping up with the real-time requirement, where the trade-off becomes more complex within dynamically evolving physical environments.

In this paper, we consider multi-modal multi-view 3D object detection in the bird's-eye view (BEV) space, as commonly adopted in autonomous driving systems. Recent works have recognized the benefits of machine perception in the BEV space [4], [5], as the object height information is less critical for driving on the road. Object detection results in the BEV space are easier to use in the following localization and planning phases. Besides, the BEV space has proved to be appropriate for merging the distance information in the LiDAR point clouds and the textual information in the RGB images for collaborative 3D object detection [6], [7]. Despite the superior performance, the complicated multi-modal multi-view collaboration incurs a high computation workload to the limited onboard processing capacity of embedded platforms.

To narrow this gap, several frameworks [8]–[11] have been proposed to prioritize the inspection of some critical regions over others. Compared to the conventional FIFO (First In First Out) paradigm that sequentially and holistically processes the sampled sensor frames, such fine-grained and priority-driven processing better accommodates the imbalanced and varying requirements on time and quality at different spatial regions. In this paper, we consider the inspection of *views* partitioned by camera-facing directions as the basic scheduling units. The problem becomes more complex due to multi-modal collaboration, as the impact of modalities on perception quality is coupled with imbalanced view criticalities. To tackle this challenge, we meticulously craft a notion of *detection utility* to depict the influence of sensor modalities and spatial views within dynamically evolving sensing contexts. Then, we formulate the scheduling problem as one maximizing the overall detection utility with deadline constraints, involving the closely coupled algorithms of *determining the resource allocation among view inspection tasks* and *effective task batching for optimizing the GPU processing throughput*.

We first observe the view inspection task is highly configurable, creating a unique space for runtime computation adaptation. Motivated by recent findings on the varying modality importance under different sensing scenarios [12], [13], we thoroughly profile the latency characteristics of a representative multi-modal 3D detector, BEVFusion [14], and define a configurable fusion space that can change the amount of processed modality data, to achieve heterogeneous accuracy-latency tradeoffs. Beyond static fusion configurations, we develop an "anytime" *elastic fusion strategy* to adaptively

*Shengzhong Liu is the corresponding author.

[1]Following the real-time conventions, we use "job" to denote the instance of periodic tasks.

979-8-3315-4026-5/24/$31.00 ©2024 IEEE
DOI 10.1109/RTSS62706.2024.00033

identify the best fusion configuration for each view with maximized detection utility, given a computation budget and the runtime sensing context (*i.e.*, object spatial distributions).

The second difference from existing works [8], [9], [11] is we consider an adaptive batching policy for *asynchronous tasks* (with mostly different release times and deadlines) with time guarantees in the FLEX. Batching is known to be effective for parallel DNN inference on GPU with improved throughput [15]. However, existing batch scheduling works [8], [10] are limited to group synchronous sub-tasks (with the same release times and deadlines) extracted from the same image, while we assume heterogeneous task periods between different views. Task batching becomes more complex because we can hesitate between waiting to group more jobs within the batch or initiating immediate execution to avoid waste. We carefully design a novel adaptive batch scheduling algorithm that intelligently groups "consecutive" view inspection jobs (ordered by a variant of non-preemptive EDF scheduling) into batches to balance resource utilization and timely job execution. The online scheduling is accelerated via an offline-constructed Hierarchical Schedule DAG (HSD) to determine the time budget of batch execution.

The two features of FLEX, *i.e.*, *elastic fusion* and *adaptive batching*, are closely integrated and mutually enhance each other. They jointly answer the questions of "what to batch" and "how to batch" during the runtime scheduling. On one hand, elastic fusion works as a subroutine to adaptive batching, automatically identifying the fusion configurations for each job to maximize a measure of detection utility, given the job set and execution time budget of the batch; On the other hand, adaptive batching effectively supports the parallel processing of view inspection jobs with heterogeneous fusion configurations generated by the elastic fusion algorithm, and it enforces temporal load balancing in earlier batch decisions to make room for future elastic fusion searches.

We implement FLEX on NVIDIA Jetson Orin and extensively evaluate its performance with a large-scale real-world driving dataset. The results show that FLEX achieves high perception quality and system throughput, outperforming the SOTA baseline by up to 22.0% in the recall, and up to 14.7% in mAP, while satisfying all timing requirements and incurring negligible runtime overhead.

Overall, the main contributions of this paper are:

- We tackle a challenging problem on real-time scheduling for multi-modal multi-view 3D machine perception, moving closer to real-world AV scenarios;
- We introduce an "anytime" elastic modality fusion strategy that dynamically decides the fusion configuration of each view given the computation time budget;
- We design a deadline-based, context-aware, and adaptive batch scheduling algorithm that combines both offline HSD construction and online selection to achieve a good trade-off between resource utilization, timely job execution, and scheduling algorithm efficiency;
- We perform extensive evaluations on a real-world driving dataset with diverse driving scenarios on Jetson Orin to validate the efficacy and efficiency of FLEX.

The rest of this paper is organized as follows: In Section II, we briefly review the related literature. We give an overview of the architecture in Section III, then explain the elastic fusion module in Section IV and the scheduling algorithm in Section V. We present the evaluation results in Section VI and conclude the paper in Section VIII.

II. Related Works

Real-time Scheduling for DNN Inference. Numerous recent studies [16]–[24] have focused on enabling real-time DNN inference on resource-constrained embedded platforms. When considering multiple parallel DNNs, existing works focused on scheduling the computation on heterogeneous processors [25], [26] and exploiting task batching techniques to improve the system throughput [27]–[30]. Besides, there have been works [8]–[11], [31] on real-time machine attention scheduling mechanisms that prioritize the inspection of regions-of-interest (ROIs) over other areas, thereby improving the criticality-aware perception quality. However, they either waste task batching opportunities on GPU [9], [11], leading to suboptimal processing throughput, or are unable to batch asynchronous tasks [8], [10] with time guarantees. Additionally, most works regard 2D object detection on RGB images as the standard perception task, and they struggle to generalize to multi-modal 3D object detection. To the best of our knowledge, no dominant task batching algorithms have been proposed for multi-modal multi-view perception with asynchronous tasks.

Resource Allocation for Multi-Modal Perception. Multi-modal 3D detector outperforms single-modal methods by effectively combining complementary information from different modalities [1], [2]. Existing real-time studies on multi-modal perception [12], [13], [32], [33] mostly worked on developing anytime inference capabilities, to maximally utilize the limited resources within the time limit, while the higher-level resource allocation among different tasks is overlooked. According to the level of processing for modality fusion, multi-modal 3D object detectors can be classified into three categories [1]: *early fusion* [34], [35] at the input data level, *middle fusion* [14], [36]–[39] at the latent feature level, and *late fusion* [40], [41] at the output decision level. In principle, FLEX can be applied to multi-modal fusion methods at different levels as long as the fusion mechanism can be calibrated for elastic modality fusion, but empirically, pushing the fusion to relatively late phases closer to the output could better accommodate dynamic DNN execution approaches, such as imprecise computation [42]–[45]. We choose the middle-level BEVFusion due to its outstanding fusion quality and practical popularity in AV literature [4], which contains deep modality feature extractors, a lightweight fusion module, and a detection head.

III. Preliminary and Overview

In this section, we first briefly introduce the preliminaries of multi-modal multi-view machine perception, and then we

Fig. 1. A multi-view multi-modal machine perception sample.

give an overview of the proposed FLEX framework.

A. Multi-Modal Multi-View Machine Perception

Mult-Modal Fusion: We consider a machine perception system for autonomous vehicles (AV) equipped with multiple sensory modalities that periodically collect signals on physical surroundings. Specifically, it includes 360-degree panoramic LiDAR and multiple cameras facing different directions. Besides, a 3D object detector (*e.g.*, BEVFusion [14]) is executed to localize and categorize surrounding objects, based on information aggregated from both LiDAR and camera input. Multi-modal fusion enhances the object detection quality but induces additional computation overhead to the limited onboard computing resources, thus presenting a trade-off between *detection quality* and *response latency*.

Mult-View Partition: The 3D object detection pipeline spatially partitions the vehicle's surroundings into multiple *views*. As shown in Figure 1, this partitioning is performed in the BEV space based on the camera-facing directions, which respectively monitor the forward, rear, left-front, right-front, left-rear, and right-rear areas of the ego-vehicle. We formally define a *view* as:

Definition 1 (View)**.** *A view is a 2D area in the* 360° *BEV space of the vehicles surroundings. Each view is associated with part of LiDAR point clouds and dedicated camera frames.*

Running 3D object detection on each view is modeled as independent periodic *view inspection tasks*, triggered at predefined *inspection frequencies*. The deadline of each view inspection job is set implicitly as the release time of the next job. Inspection frequencies are different but static among the views, according to their heuristic importance to the ego-vehicle's safety. For example, the front-view area is generally more important than the left/right-rear-view areas and should be inspected more frequently. The sampling frequencies of LiDAR and cameras can be different, both of which are no smaller than the highest view inspection frequency. Upon the release of a new view inspection job, the perception pipeline collects the most up-to-date data from both modalities using a sensor synchronization mechanism, such as SEAM [46], to prepare the input for 3D detection.

The limited computation resources must be appropriately allocated among different view inspection tasks for responsiveness to physical surroundings. Moreover, as the AV navigates, the criticality of different views may dynamically change due

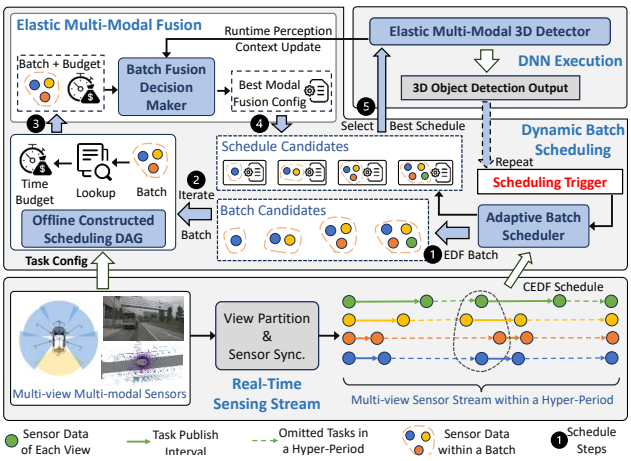


Fig. 2. Overview of the FLEX framework.

to evolving driving conditions (*i.e.*, object location distributions) and runtime vehicle maneuvers (*e.g.*, turn left/right, accelerate/decelerate). We fix the view inspection frequencies but dynamically change the amount of computation at each view to achieve adaptive resource allocation and sustain consistently high 3D object detection quality.

B. Motivation and System Overview

The efficiency and efficacy requirements for machine perception motivate us to raise the following three questions:

- **Q1:** How to adaptively and intelligently allocate computation resources among different view inspection tasks to maximize the overall perception quality?
- **Q2:** How to fully utilize the parallel processing capacity of the GPU processor to improve the system throughput and facilitate the perception quality?
- **Q3:** How to ensure the timely execution of all view inspection tasks on different views?

To answer these questions, we propose FLEX, a real-time scheduling framework for multi-modal multi-view machine perception, that integrates *elastic fusion* and *adaptive batching* as two key features, to appropriately allocate machine attention (*i.e.*, GPU computation time) [10] among spatial views and sensory modalities. We now give intuitions on how these features jointly achieve the design objectives.

- **Elastic Fusion:** It serves as an "anytime" solution for the multi-modal object detector and is able to produce available detection results within any feasible time limit (**Q3**). It adaptively adjusts the amount of processed LiDAR and camera input within each inspection task, judiciously compressing computation on less-critical views to make room for critical view inspections (**Q1**).
- **Adaptive Batching:** It groups different view inspection jobs into a single GPU request for improved task processing throughput (**Q2**). We consider a flexible setting that allows batching jobs with (1) asynchronous release times and (2) heterogeneous multi-modal fusion configurations.

The adaptive batch decisions naturally fit the purpose of dynamic resource allocation (**Q1**).

The remaining question is: how to design an **adaptive batch scheduling policy** that simultaneously satisfies requirements **Q2** and **Q3**? Our intuition is that *jobs with similar deadlines are more suitable to form a batch* because batched jobs are enforced to share the same finish time. Batching jobs with diverse deadlines may cause a lack of sufficient execution duration for jobs with later deadlines. Based on this observation, we develop a deadline-driven batch scheduling algorithm that groups "consecutive" jobs based on a non-preemptive EDF job sequence. Besides, it carefully determines the time budget of a batch to ensure no deadlines are missed for both currently active and future unpublished jobs, which is efficiently done with the assistance of an offline-constructed hierarchical schedule DAG (HSD).

An overview of the proposed FLEX architecture is presented in Figure 2. To effectively combine the design features, FLEX comprises the following building components:

- **Batch Fusion Decision Maker:** Given a set of view inspection jobs and the corresponding batch execution budget, it decides feasible fusion configurations for each job that maximizes an overall detection utility measure.
- **Elastic Multi-Modal 3D Detector:** It accepts and processes adjustable multi-modal sensory data from different view inspection jobs and generates detection results.
- **Offline Hierarchical Schedule DAG:** It serves as a fast lookup table for the adaptive batch scheduler to determine the latest finish time of a batch's execution.
- **Adaptive Batch Scheduler:** It iterates potential candidate batches based on the Earliest Deadline First (EDF) order and selects the one yielding the highest expected detection utility gain to execute.

IV. Object Detection With Elastic Fusion

In this section, we introduce the elastic multi-modal 3D detector and explain how we determine the runtime fusion strategy to maximize the notion of detection utility.

A. BEVFusion for Multi-Modal 3D Object Detection

BEVFusion [14] is an efficient and generic multi-modal multi-task framework. It represents and fuses LiDAR and image features in the unified BEV space, and can be applied to a variety of perception tasks, *e.g.*, 3D object detection and map segmentation. We regard 3D object detection as the standard machine perception task and provide a brief overview of the BEVFusion-based object detection network (hereafter referred to as BEVFusion without ambiguity). A multi-view BEVFusion detection pipeline is demonstrated in Figure 3, and the detector comprises the following four modules:

- **LiDAR Feature Extractor:** It first voxelizes the raw LiDAR point clouds and then performs sparse convolutions on voxels to extract features in the BEV space.
- **Image Feature Extractor:** It feeds images into 2D backbones and then projects 2D features into BEV space via view transformation and BEV pooling.

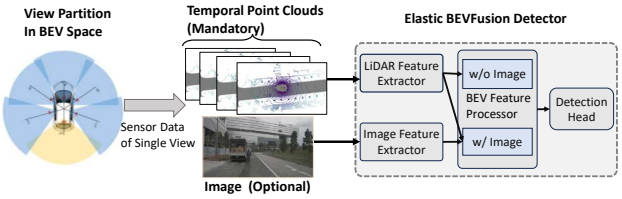


Fig. 3. Multi-view Multi-modal BEVFusion Detection Pipeline.

- **BEV-Feature Processor:** It processes the extracted features in the shared BEV space, where the LiDAR features are always required but the image features can be optional. When both are available, the module concatenates them, conducts convolutions for modality fusion, and generates the fused features in the BEV space.
- **Detection Head:** It utilizes the BEV features to predict the category, location, and size of the appeared objects. An oriented 3D bounding box and a class are predicted for each object.

Different modules of BEVFusion run sequentially on GPU, but the view inspection jobs can be batched within each module.

LiDAR and Image Fusion: In 3D object detection, the depth information of point clouds is critical for accurate object localization, while the high-resolution image textual information serves as its complement and can be skipped under tight resource constraints.

Temporal Point Cloud Fusion: A common practice to improve LiDAR-based 3D object detection quality is to merge point clouds in past scans, for the enhanced description of object location and structure. The points collected in past scans are appended to current LiDAR points and fed into the LiDAR feature extractor, where newly collected points possess higher weights than history points. In FLEX, we exploit timestamp-aware multi-frame point cloud concatenation for temporal fusion. Upon dynamic resource availability, different numbers of LiDAR frames can be fused to achieve elastic accuracy-latency trade-offs. Empirically, there is an upper bound K of LiDAR frames below which fusing more frames leads to higher detection quality [47].

B. Batch Execution of View Inspection Jobs

Batch execution processing multiple data samples in parallel on GPU improves the DNN inference throughput compared to sequential execution [15]. Here we introduce the proposed batch execution model that allows the batched view inspection jobs to be configured differently during execution.

Assume the view set is $\{r_i\}_1^n$. The view inspection task is highly configurable in our tailored BEVFusion implementation. The *fusion configuration* of a single view r_i is (l_i, b_i), where $l_i \geq 1$ is the number of LiDAR frames for temporal fusion, and $b_i \in \{0, 1\}$ is a binary variable indicating whether to incorporate the image frame. Since l_i only affects the point list length and b_i only affects the number of processed images, view inspection jobs with different fusion configurations can be directly batched within each BEVFusion module. Only a single forward pass is needed in each feature extractor.

Algorithm 1: Batch Fusion Configuration Search

Input: Views $\{r_j\}$ to inspect, time budget T, driving context represented by $\{w_j\}$.
Output: Fusion configuration for each view $\{(l_j, b_j)\}$.

1 Determine the maximum number of fused images;
2 **for** *each feasible image configuration* $\{b_j\}$ **do**
3 Obtain optimal fractional solution $\{\hat{l}_j\}$ using the Lagrange multiplier method;
4 Convert $\{\hat{l}_j\}$ to feasible integer solution $\{l_j\}$ greedily;
5 **end**
6 Return configuration $\{(l_j^*, b_j^*)\}$ with the maximum utility among all configuration candidates.

We now analyze the worst-case execution time (WCET) of batch detection for a view subset $\{r_j\}$, where $|\{r_j\}| \geq 1$. The WCET of BEVFusion is decomposed into its four modules:

- The LiDAR branch execution time mainly depends on the point count in the point cloud, nearly proportional to the LiDAR frame number used in all views, so we use $t_l(\sum_j l_j)$ to denote the WCET of the LiDAR branch.
- The image branch execution time is related to the number of batched images and its WCET is denoted as $t_i(\sum_j b_j)$.
- The execution time of the BEV-feature processor depends on whether image features are incorporated. Therefore, its WCET can be expressed as a binary-value function $t_p(B)$, where B indicates whether $\sum_j b_j > 0$ holds.
- The execution time of the detection head is roughly constant, denoted as t_h.

Therefore, the batch execution's WCET for a set of view inspection jobs with configurations $\{(l_j, b_j)\}$ is calculated by:

$$\text{WCET} = t_l(\sum_j l_j) + t_i(\sum_j b_j) + t_p(B) + t_h. \tag{1}$$

C. Elastic Fusion Configurations Search

We now explain the proposed elastic modality fusion strategy. Given a set of view inspection jobs in the batch and the associated execution time budget, the problem is formulated as determining the feasible fusion configuration for each job to maximize a notion of *utility* without violating the time limit.

The utility provides a proxy measure of perceptual quality that considers both driving contexts and fusion configurations. It should be calculated without ground-truth labels to be effective during runtime adaptation.

Definition 2 (Detection Utility)**.** *The detection utility of inspecting view r_i with configuration (l_i, b_i) is defined as:*

$$u_i(l_i, b_i) = w_i \cdot (1 + \alpha_i b_i) \cdot f(l_i). \tag{2}$$

The factors contained in the utility are listed below:

- $\mathbf{w_i}$ *is the ratio of objects in view r_i compared to the total object number in all views, as counted from the most recent inspection*[2].
- $\alpha_\mathbf{i}$ *is the relative detection quality gain ratio when fusing image features compared to LiDAR-only approaches, measured through offline profiling.*
- $\mathbf{f(l_i)} = \log l_i$ *is the relative detection quality ratio achieved through temporal point cloud fusion.*

w_i is utilized as a runtime driving-context variable decided by object location distributions. It is noteworthy that we select $f(l_i)$ as a monotonically increasing concave function, as is consistent with our offline profiling results in Figure 6.

Given a batched view set $\{r_j\}$ and a time budget T, the optimal configuration search for each view is mathematically formulated as an integer programming problem to maximize the overall detection utility:

$$\max_{\{(l_j, b_j)\}} \sum_j u_j(l_j, b_j) \tag{3a}$$

$$\textbf{s.t.} \quad t_l(\sum_j l_j) + t_i(\sum_j b_j) + t_p(B) + t_h \leq T \tag{3b}$$

$$l_j \in \{1, \ldots, l_{max}\},\ b_j \in \{0, 1\} \tag{3c}$$

Constraint (3b) guarantees that the time budget is not exceeded. Note that if a specific view is chosen for inspection, at least one corresponding LiDAR frame will be processed.

We present our batch fusion configuration search algorithm of FLEX in **Algorithm 1**, which returns approximately optimal configurations. Here, "approximately optimal" means that we first find the optimal fractional solution and then approximate it to a feasible integer solution. We only assign a single batch for LiDAR and image feature extraction separately[3], and then figure out "what to include in the predefined LiDAR batch and image batch". Iterating over all feasible configurations $\{(l_j, b_j)\}$ is too time-consuming, so we design a two-step method that first enumerates all possible image configurations $\{b_j\}$ (in Lines 1 and 2). Once the first-step decision is made, we determine the LiDAR frame number for each view by solving a convex optimization problem without integer constraints, *i.e.*,

$$\max_{\{l_j\}} \sum_j wl_j \cdot f(l_j) \tag{4a}$$

$$\textbf{s.t.} \quad t_l(\sum_j l_j) \leq T_l \tag{4b}$$

Here $wl_j = w_j \cdot (1 + \alpha_j b_j)$, and T_l is the time budget for LiDAR feature processing after subtracting the WCET of other modules from T. We use the Lagrange multiplier method [48] to solve this problem. First, assuming that $t_l(L) = T_l$ holds, then the maximum overall utility is obtained when $\sum_j l_j = L$ holds since $f(l_j)$ is monotonically increasing. Second, by leveraging the optimality condition of the Lagrange multiplier method and the fact that $f(l_j)$ is concave, we have that at the optimal solution $\{\hat{l}_j\}$, the derivatives of all utility functions $wl_j \cdot f(\hat{l}_j)$ for all views are the same. Based on the above two

[2] w_i can be extended to consider the heterogeneous object importance or driving behaviors, but more complicated w_i designs are beyond our scope.

[3] In our context with limited batch candidates, for both LiDAR and image feature extractors, initiating a second batch is always less efficient than adding the data to the original first batch.

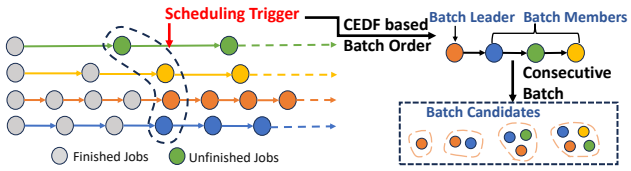


Fig. 4. Task publish model and EDF-DB scheduling.

equations, we can quickly solve the optimal fractional solution $\{\hat{l}_j\}$.

We then approximate $\{\hat{l}_j\}$ to a feasible integer solution. Specifically, we sum up all decimal parts of $\{\hat{l}_j\}$ (including parts that exceed l_{max}) and greedily reallocate these spare LiDAR frames sequentially to the view that yields the maximum utility gain. The algorithm finally returns integer configurations $\{(l_j, b_j)\}$ that achieves the maximum overall utility among all the configuration candidates where each configuration candidate corresponds to a feasible image configuration.

V. Adaptive Batch Scheduling

In this section, we introduce the EDF-DB (Earliest Deadline First scheduling with Dynamic Batching) scheduler in FLEX, a non-preemptive deadline-based scheduling algorithm that enables dynamic *online batching* and *job configuration*. Generally, it adaptively decides what to batch and how to batch in three steps: (1) Generate batch candidates based on the job deadlines. (2) Determine the batch execution time budget with the help of offline-constructed HSD. (3) Invoke the batch fusion configuration search and select the optimal batch candidate to execute.

A. Execution Model and Problem Formulation

Task Execution Model: We regard the multi-modal 3D object detection at different views as a collection of periodic tasks $\{\tau_i\}_1^n$, with the period of τ_i as P_i. We use $\tau_{i,j}$ to denote the j-th job of τ_i and use $r_{i,j}$ to denote its release time, and we assume the initial jobs for all tasks are released synchronously. At most one job of a task is executed non-preemptively on the GPU at any time.

Execution Time Notations: As detailed in section IV-B, the end-to-end WCET of task τ_i is a function of runtime fusion configuration (i.e., whether to fuse image features and the number of LiDAR frames). The WCETs and BCETs of the most lightweight configuration, *i.e.*, when only one frame of LiDAR point cloud is used without incorporating image features, are denoted with $\{c_i^{min}\}, \{bc_i^{min}\}$, respectively.

Scheduling Problem Formulation: When the GPU becomes idle after a previous batch or when a new job is published during the processor's idle time, the adaptive batch scheduling algorithm should be invoked to determine for the next batch, (1) which jobs are grouped in the batch and (2) the fusion configuration for each job. We give a formal batch definition in the scheduling context.

Definition 3 (Scheduled Batch)**.** *A scheduled batch is associated with four elements* $(\{\tau_{i_k,j_k}\},\ \{(l_k,\ b_k)\},\ s,\ d)$ *where:*

- $\{\tau_{i_k,j_k}\}$ *are the jobs within the batch.*
- $\{(l_k, b_k)\}$ *denotes the fusion configuration for each job.*
- s, d *are the start and end times of batch execution.*

Given the task set, the scheduling algorithm adaptively generates a sequence of batches one by one to maximize the batch detection utility with no deadline misses.

B. Batching Order Based on Non-Preemptive EDF

Following the insight to group jobs with similar deadlines into batches, we first sort the jobs according to an EDF order and use it as the scheduling basis for our adaptive batch scheduling. Particularly, we adopt a non-work-conserving EDF algorithm called CEDF (Clairvoyant non-preemptive EDF) [49], that intelligently inserts idle times to prevent upcoming priority inversion that will cause a deadline miss. Specifically, when a job is scheduled according to the EDF order, CEDF will check jobs that it may block, i.e., those jobs with earlier deadlines but later release times, and check whether this priority inversion leads to a deadline miss. We give the following property of CEDF scheduling for a periodic task set w.r.t. their initial job release times.

Lemma 1. *For a set of jobs generated by a concrete periodic task set, if it is schedulable under CEDF starting at time t, then it is schedulable under CEDF starting at time t', for all $t' < t$.*

Proof. We prove the lemma by contradiction. Assume there exist t_1 and t_2 such that $t_1 < t_2$. Under CEDF scheduling, the job set is schedulable starting at t_2 but not schedulable starting at t_1. For the schedule beginning at t_1, let the first job that misses the deadline be j_L, with a deadline of d_L.

Consider the sets of completed jobs scheduled by CEDF up to d_L from the two starting points, denoted as J_1 and J_2, respectively. J_1 can't be a proper subset of J_2, otherwise, the schedule starting at t_1 idles while there exist ready jobs whose execution will not lead to a deadline miss because $t_1 < t_2$ and the schedule starting at t_2 is feasible. This means J_1 contains a job j_E that doesn't belong to J_2. Note that the deadline d_E of j_E must be later than d_L otherwise the schedule starting at t_2 also misses this deadline. Therefore, in the schedule starting at t_1, a priority inversion between j_L and j_E that causes a deadline miss happens, which will not happen under CEDF scheduling. □

We further define the hyper-period for a task set:

Definition 4 (Hyper-Period)**.** *The hyper-period HP of a task set is defined as the least common multiple of all task periods, such that*

$$HP = HP(\{\tau_i\}_1^n) = lcm(P_1, \ldots, P_n), \tag{5}$$

Without loss of generality, we assume all tasks start at time 0. Thus, to evaluate the schedulability of the task set with given task WCETs $\{c_i\}$ using CEDF, we only need to check whether the task set is schedulable in the first hyper-period from 0 to HP. Besides, we assume the task

Algorithm 2: The EDF-DB Scheduling Algorithm

Input: Task set with periods $\{P_i\}$ and WCETs $\{c_i^{min}\}$, current time t.
Output: A feasible batch with maximized detection quality.

```
Select job τ_{a1,b1} according to CEDF;
Determine batch member sequence τ_{a2,b2}, ..., τ_{an,bn} that
  may batch with τ_{a1,b1} according to the order under CEDF
  scheduling;
for each batch candidate do
    Determine the latest finish time lft with the help of
      HSD ;
    Calculate the actual start and finish time s, d for this
      batch given by formula (6) ;
    Determine the actual configuration for the batch, and
      calculate the utility gain ;
end
Return the batch candidate with the highest average utility
  gain and the corresponding configurations.
```

set is always schedulable with CEDF scheduling under their most lightweight configurations. Although determining the schedulability of concrete non-preemptive periodic tasks is NP-Hard in a strong sense [50], by confining ourselves to special cases of a concrete periodic task set with synchronized starting time, we can quickly run an offline simulation to evaluate its schedulability within a hyper-period.

C. EDF-DB: Adaptive Batch Scheduling

We now introduce an EDF-DB algorithm in FLEX for adaptive batch scheduling. Since we only allow consecutive jobs within a batch, the search space for batches on synchronized tasks is much smaller. Our general idea is to dynamically group appropriate jobs and determine the time budget of these jobs to form a *feasible batch*. The feasibility of a scheduled batch is defined below:

Definition 5 (Feasible Batch)**.** *A scheduled batch* $(\{\tau_{i_k,j_k}\},\ \{(l_k,\ b_k)\},\ s,\ d)$ *is* feasible *if it satisfies:*

1) *The release time of any job* τ_{i_k,j_k} *is no later than the batch start time* s.
2) *The absolute deadline of any job* τ_{i_k,j_k} *is no earlier than the batch end time* d.
3) *The execution time of the selected configuration* $\{(l_k, b_k)\}$ *is no longer than the assigned execution time* $d - s$.
4) *The remaining jobs in the same hyper-period are schedulable with CEDF under their most lightweight configurations* $\{c_i^{min}\}$ *starting from time* d.

In Condition 4), when considering the current batch decision, we ensure the schedulability of remaining jobs within the same hyper-period under sequential execution. This design preserves time budgets for potential configuration upgrades when future jobs are executed in batch, leading to temporal load balancing and consistent detection quality over time.

When multiple feasible batch candidates exist, EDF-DB selects the one with the maximum expected detection quality gain. Within each batch, EDF-DB considers the current detection context (*i.e.*, object spatial distributions) and determines $\{(l_k, b_k)\}$ for jobs in the batch satisfying condition 3) to maximize the batch detection quality.

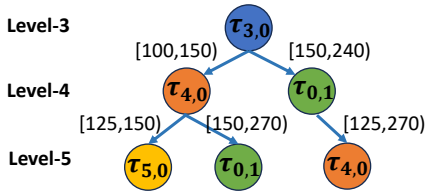


Fig. 5. An example of the intermediate 3 layers of HSD. For instance, if the level-3 node's corresponding job $\tau_{3,0}$ finishes during the time interval [100, 150), then the next job scheduled by CEDF is $\tau_{4,0}$.

The EDF-DB algorithm is summarized in **Algorithm 2**. CEDF may insert idle times to prevent upcoming deadline misses. In such cases, our EDF-DB also keeps the processor idle until this idle time finishes; otherwise, the deadline miss is bound to happen. When EDF-DB is invoked at time t, it first selects the job τ_{a_1,b_1} according to CEDF, which we call the *batch leader*. Then we consider the sequence of jobs (batch members) $\tau_{a_2,b_2}, \ldots, \tau_{a_n,b_n}$ that may join this batch according to the CEDF scheduling order, assuming job execution times are $\{c_i^{min}\}$. We only need to consider jobs whose release time is before the batch leader's deadline. Some jobs in the batch may have not been released at time t, which means the batch needs to wait until their releases before executing.

As mentioned, EDF-DB follows the order given by CEDF scheduling. As shown in Figure 4, only batches that group consecutive jobs, conforming to the form $\{\tau_{a_1,b_1}, \tau_{a_2,b_2}, \ldots, \tau_{a_q,b_q}\}$, are allowed. In other words, no intermediate jobs can be skipped within the CEDF job sequence. Such a potential feasible batch is called a *batch candidate*. For each batch candidate, we first determine its latest finish time (lft), which ensures all jobs after this batch will meet their deadlines under the original CEDF scheduling. This is calculated by our HSD algorithm (as introduced later), and the lft of the current batch equals to the latest start time of future jobs. Assume the latest release time and the earliest deadline within the batch are respectively r_{max}, d_{min}, then the actual start and end time of the batch's execution are given by:

$$s = \max(r_{max},\ t), \quad d = \min(d_{min},\ lft). \tag{6}$$

Once we determine the actual time budget for a candidate batch (*i.e.*, what to batch), we can accordingly search its best runtime job configurations using **Algorithm 1** (*i.e.*, how to batch), and calculate the batch detection utility. Then, we choose the batch candidate with the highest average utility gain, along with its best job configurations, for execution.

D. Hierarchical Schedule DAG

Within EDF-DB scheduler execution, repetitively analyzing the CEDF schedulability of remaining jobs in the same hyper-period for each candidate batch can be time-consuming. Fortunately, the schedulability under the *most lightweight configurations* is consistent across different hyper-periods. To accelerate the online scheduling, we introduce an offline algorithm to

Algorithm 3: Hierarchical Schedule DAG Construct

Input: Task set τ with periods $\{P_i\}$, WCETs $\{c_i^{min}\}$, BCETs $\{bc_i^{min}\}$.
Output: A hierarchical schedule DAG $G(V, E)$.

1 Initialize $G(V, E)$ as empty graph;
2 Select the job $\tau_{m,0}$ with the earliest deadline among all jobs, add $\tau_{m,0}$ to V;
3 $N :=$ the number of total jobs in a hyper-period;
4 **for** $k = 1, \ldots, N-1$ **do**
5 V_{k-1} := set of nodes whose level is $k-1$;
6 $(V_{k-1,1}, \ldots, V_{k-1,p}) :=$ partition of V_{k-1} based on finished job set of each node ;
7 **for** $l = 1, \ldots, p$ **do**
8 $JR_{k-1,l} :=$ remaining job set of $V_{k-1,l}$;
9 $est :=$ earliest start time of $JR_{k-1,l}$;
10 $\{j_o\}, \{[left_o, right_o)\} :=$ the set of first scheduled job under CEDF and its corresponding start interval computed by **Algorithm 4**;
11 Add each job j_o in $\{j_o\}$ to V as a level-k node v_o ;
12 **for** $\forall u \in V_{k-1,l}$, $\forall v_o \in \{v_o\}$ **do**
13 if condition (9) holds, then add to E an edge from u to v_o, the assigned interval of the edge is calculated by formula (10);
14 **end**
15 **end**
16 **end**
17 Return the constructed $G(V, E)$.

Algorithm 4: First Job Enumeration (FJE)

Input: remaining job set JR, earliest start time est.
Output: possible first jobs scheduled by CEDF and the corresponding start time intervals $\{j_o\}, \{[left_o, right_o)\}$.

1 $fd :=$ the first deadline among all jobs in JR ;
2 Binary search to find the latest start time $lst \in [est, fd]$;
3 divide $[est, lst)$ into adjacent intervals $\{[left_o, right_o)\}$ based on the first scheduled job j_o under CEDF ;
4 Return $\{j_o\}, \{[left_o, right_o)\}$;

construct a *Hierarchical Schedule DAG (HSD)*. It covers all possible CEDF job execution sequences within a hyper-period, used as a "fast lookup table" to efficiently determine the execution time budget of all candidate batches by looking up the latest start time (*lst*) of all possible remaining job sets without complex runtime profiling.

Figure 5 visually illustrates a part of HSD. For a constructed HSD $G(V, E)$, each node $v \in V$ corresponds to a job, and the directed edges represent the order of job executions. HSD is hierarchical:

1) It has a single *level*-0 root node with no in-edge, and the level of node v is the distance from the root to v;
2) An edge exists only if it points from a level-k node to a level-$(k+1)$ node. Any path starting with the root in HSD represents a sequence of job execution under CEDF scheduling.
3) An edge $e = (u, v) \in E$, *i.e.*, pointed from u to v, is assigned with a time interval, which is formally defined below.

Definition 6 (Edge Time Interval)**.** *An edge (u, v) is assigned with the left-closed, right-open time interval $[s, t)$, so that if the sequence of jobs from the root to u finishes their execution within $[s, t)$, then the next job scheduled by CEDF is v.*

Formally, given a start-aligned task set, the pseudocode of constructing its HSD in a hyper-period is summarized in **Algorithm 3** (**HSDC**). HSDC starts by adding the job with the earliest deadline as the root (Line 2). Assume the number of jobs within a hyper-period is N (Line 3), then the maximum level in the graph is $N-1$. The graph is constructed level by level, as shown between Lines 4 to 17.

In the k-th iteration for graph construction, HSDC first partitions all level-$(k-1)$ nodes according to their *finished job set* (Line 5 to 6), *i.e.*, two nodes are divided into the same *partition* if and only if they have the same finished job set. The finished job set of a node v, denoted by $J(v)$, contains jobs in the path from the root to node v. Though there could be multiple paths from the root to the node v, as we will show later in Lemma 2, their job sets are the same. Thus, the finished job set of a node is well-defined, and the same with the partition based on it. For each partition $V_{k-1,l}$ of the node-set at level-$(k-1)$, we construct new nodes at level-k and add edges from nodes in $V_{k-1,l}$ to the newly added nodes, as shown between Lines 8 to 14.

For partition $V_{k-1,l}$ and its finished job set $J_{k-1,l}$, its *remaining job set* is defined as $JR_{k-1,l} = J - J_{k-1,l}$, where J is the set of all jobs within a hyper-period (Line 8). Then we calculate the possible earliest start time (est) of $JR_{k-1,l}$ by the following equation:

$$est(JR_{k-1,l}) = \min\{eft(v) \mid v \in V_{k-1,l}\}. \tag{7}$$

Here $eft(v)$ is the earliest finish time of the job by v. The earliest finish time of the root $\tau_{m,0}$ is bc_m^{off}. For non-root node v representing job $\tau_{i,j}$, suppose the set of time intervals assigned with edges pointing to v is $\{[s_m, t_m)\}$, then $eft(v)$ is calculated by:

$$eft(v) = \min\{\max(s_m, r_{i,j}) + bc_i^{off}\}. \tag{8}$$

Recall that $r_{i,j}$ is the release time of $\tau_{i,j}$.

HSDC then invokes a **First Job Enumeration (FJE)** subroutine (in **Algorithm 4**) with the remaining job set and the earliest start time (Line 10). It first uses binary search to find JR's latest start time (lst) (Lines 1, 2), *i.e.*, JR is schedulable with the execution time $\{c_i^{min}\}$ under CEDF only if starting before lst. The binary search's correctness is guaranteed by the CEDF property stated in Lemma 1. Then, FJE divides the feasible start time interval $[est, lst)$ into a set of disjoint adjacent time sub-intervals $\{[left_o, right_o)\}$ according to the first job j_o of JR scheduled by CEDF starting within the sub-interval. After FJE returns all possible first jobs $\{j_o\}$ for the target remaining job set, HSDC adds each job j_o as a new level-k node v_o to the node set V (Line 11). Then HSDC adds edges from a node u in $V_{k-1,l}$ to a newly added node v_o (Lines 12 to 14) if the following equation holds:

$$eft(u) < right_o. \tag{9}$$

The assigned interval of the newly added edge is given by:

$$[\max\left(left_o, eft(u)\right), \quad right_o). \tag{10}$$

We then show that the finished job set $J(v)$ of a node v in HSD is well-defined.

Lemma 2. *For a node v in a constructed HSD, the set of jobs on any path from the root to v are the same.*

Proof. We prove the lemma by induction. It holds for the root of the HSD, whose finished job set contains only itself. Assume it holds for all level-k nodes. Then for any level-$(k+1)$ node v, we consider the set of level-k nodes that have an edge pointing to v, denoted by $P(v)$. Our construction process guarantees that all nodes in $P(v)$ belong to the same partition of the level-k node set. Applying the inductive hypothesis, the finished job sets of nodes in $P(v)$ are unique and identical. Let U be the finished job set corresponding to this partition. We then have that the set of jobs on any path from the root to v must be $U + \{v\}$, which proves the lemma. □

How to use the constructed HSD to determine the latest finish time (lft) for each batch candidate in EDF-DB? For a batch, assume the finished job set (including the batch itself) is J, and the last job in this batch (according to CEDF order) is j. Then we locate the corresponding node v representing job j whose finished job set is J in the $(|J|-1)$-th level of the HSD. Assume the set of time intervals assigned to the out-edges of v is $\{[s_q, t_q]\}$, then the latest finish time of the target batch is given by $lft = \max\{t_q\}$.

Space complexity of HSD. Assuming the number of all jobs in a hyper-period is N, and the maximum number of nodes in a level of HSD is M. Due to the hierarchy of HSD, the required space to store the nodes and edges of HSD are $O(NM)$ and $O(NM^2)$ respectively. In our evaluation which typically involves hundreds of jobs in a hyper-period, M is usually smaller than 15. The total space required to store the HSD typically ranges from tens to hundreds of kilobytes, which is acceptable on our test embedded device (see Section VI-A).

VI. Evaluation

In this section, we evaluate the effectiveness and efficiency of FLEX on the NVIDIA Jetson Orin platform with a large-scale real-world driving dataset.

A. Experimental Setup

1) Hardware Platform: All experiments are conducted on an NVIDIA Jetson Orin SoC, which is designed for automotive embedded systems. It is equipped with a 12-core Arm Cortex-A78AE v8.2 64-bit CPU, a 2048-core NVIDIA Ampere architecture GPU with 64 Tensor Cores, and 64 GB memory. We set the power mode to MAXN and set the GPU and CPU to operate at their maximum frequency to ensure stable performance.

2) Dataset: We test on the nuScenes [51] dataset, a large-scale public dataset for autonomous driving developed by the team at Motional. It consists of 20-second driving video clips collected in Boston and Singapore captured by various sensors. We use the sensor data from 6 cameras and the LiDAR mounted on the top. The cameras are positioned to face forward, rearward, left-front, right-front, left-rear, and right-rear directions. We conducted tests using **85 driving scenes**, comprising over 3400 data samples[4]. Each sample includes a LiDAR point cloud frame and 6 images.

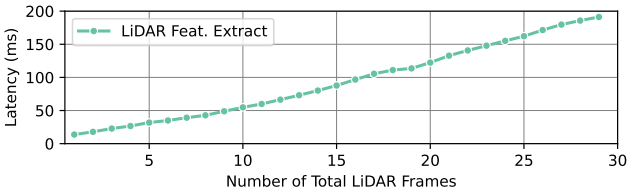
3) Neural Network for Detection: We use the BEVFusion [14] model in MMDetecion3D [52] library as the 3D object detection network. We calibrate its pipeline to support elastic multi-modal fusion. We use TensorRT with FP16 precision to speed up the image feature extractor and detection head. The LiDAR feature extractor is accelerated with the fast 3D sparse convolution library [53] published by NVIDIA with FP16 precision. The worst-case inference latency of different modules is profiled in advance.

4) Workload Setup: Unless otherwise indicated, we regard the inspection of 6 views, as partitioned by the camera facing directions, as 6 periodic tasks with implicit deadlines. We manually change the task periods to induce diverse workloads. Under our implicit deadline model, shorter periods lead to higher scheduling workloads. Specifically, we assign a period of 160ms, 200ms, 250ms, 300ms, 400ms, and 600ms to the 6 tasks to represent a moderate workload. To create the easy workload, we multiply each period by 1.1; for the hard workload, we multiply them by 0.9. Specifically, the task set's utilization (*i.e.*, sum of the ratio between WCET and the period of all tasks) with the minimum fusion configuration (*i.e.*, only one LiDAR frame is used without incorporating image features) for easy, moderate, and hard workloads 72.4%, 79.6%, and 88.5%, respectively.

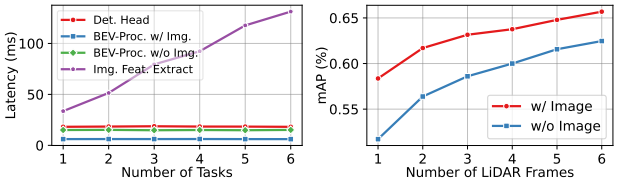
5) Evaluation Metrics: Our metrics mainly consider the overall 3D object detection quality. We use detection recall, precision, and mean average precision (mAP) to evaluate the detection accuracy. These metrics are calculated based on the bounding box overlap between the predictions and the groundtruth labels. A detection is said to be matched with a groundtruth object if their intersection of union (IoU) is larger than a predefined threshold (set as 0.35 in this paper). We also evaluate the effectiveness of FLEX in utilizing computational resources, measured by the ratio of processed sensor data over the maximum configuration, which includes 5 history LiDAR frames as well as images from all spatial views. Specifically, the following metrics are defined:

- **Detection Recall:** The ratio between detections matched with groundtruth objects and the total groundtruth objects.
- **Detection Precision:** The ratio between detections matched with groundtruth objects and the total detected objects.

[4]In nuScenes, the labeling frequency is 2Hz, and the frequencies of LiDAR and cameras sampling are 20Hz and 12Hz, respectively. Unlabeled LiDAR frames are used for temporal point cloud fusion.



(a) Latency of LiDAR Branch.



(b) Latency Breakdown. (c) Accuracy Profile.

Fig. 6. Accuracy and latency profiles on different fusion configurations.

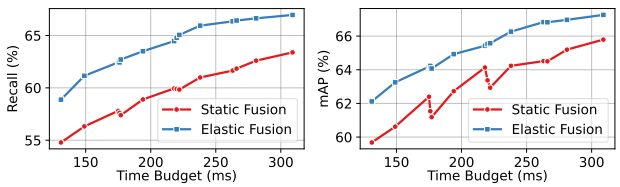
- **Mean Average Precision (mAP):** The mean of the average precision among all classes. It provides a comprehensive assessment of the model's overall performance in detecting objects across different classes.
- **LiDAR/Image Process Ratio:** The ratio between the processed LiDAR/image frames and the maximum configuration volume.

B. Offline Latency and Accuracy Profiling

Different multi-modal fusion configurations can lead to a wide spectrum of inference latency and detection accuracy. Here we investigate their trade-off and report a breakdown of batch execution latency for all modules under different modal fusion configurations. The obtained profiling results are reported in Figure 6. We have the following observations: First, both incorporating image features and increasing the number of LiDAR frames in temporal fusion improve the detection quality. Different configurations may achieve similar detection accuracy but vary significantly in inference latency, thus requiring careful runtime selection. Second, all BEV-Fusion modules can benefit from batched execution. As the number of batched tasks increases, the execution time for the detection head and BEV-feature processor remains relatively stable. On the contrary, the latency of LiDAR and image feature extractors increases along with the increasing input data volume, but it is still more efficient than sequential task execution.

C. Contribution of Elastic Fusion

We compare the context-aware elastic fusion strategy in FLEX with the static fusion strategies. Given a static fusion configuration, we determine its WCET via offline profiling, which is further used as the time budget for our elastic fusion algorithm. The obtained mAP and recall results are reported in Figure 7. The results show that the elastic fusion consistently outperforms the static fusion strategy with a clear margin on the achieved mAP and recall. With the same time budget, elastic fusion can appropriately allocate the computation resources to critical views that can receive higher detection utility and



(a) Detection recall. (b) mAP.

Fig. 7. Comparison of elastic fusion and static fusion on detection accuracy.

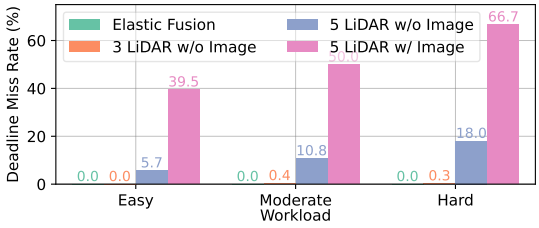


Fig. 8. The deadline miss rate of elastic fusion and static fusion.

TABLE I
ACCURACY COMPARISON BETWEEN ELASTIC FUSION AND STATIC FUSION CONFIGURATION OF 3 LIDAR FRAMES WITHOUT IMAGE FUSION.

Fusion Strategy	Elastic (Easy)	Elastic (Moderate)	Elastic (Hard)	Static
Recall	60.8%	59.7%	58.1%	52.3%
mAP	63.9%	63.4%	62.4%	58.6%

adaptively select suitable configurations (*i.e.*, history LiDAR frames and whether to use the image frame), both of which benefit the overall detection quality. Instead, static fusion strategies lack flexibility in view configuration selection, and they may suffer in scenarios where objects are unevenly distributed between spatial views.

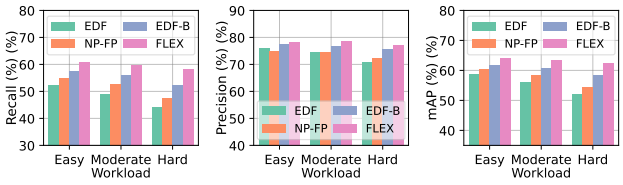
We further prove the schedulability improvement brought by the elastic fusion through ablation studies. We compare the job deadline miss rate of elastic fusion and static fusion under three workloads. We employ the same batch scheduling policy proposed in Section V on all fusion strategies for a fair comparison. The results are shown in Figure 8. Our elastic multi-modal fusion detector serves as an "anytime" neural network, *i.e.*, it can yield available detection results given different time budgets, leading to no deadline misses. However, static fusion suffers from severe deadline misses when adopting computation-intensive configurations, *e.g.*, 5 LiDAR frames with image fusion. Although conservative fusion configurations, *e.g.*, 3 LiDAR frames without image fusion, can alleviate deadline misses, as shown in Table I, they fail to fully utilize computational resources and cause degraded detection quality.

D. Scheduling Algorithm Comparison

In this subsection, we evaluate the scheduling performance of FLEX compared to several baselines, under different scheduling workloads (*i.e.*, easy, moderate, and hard load).

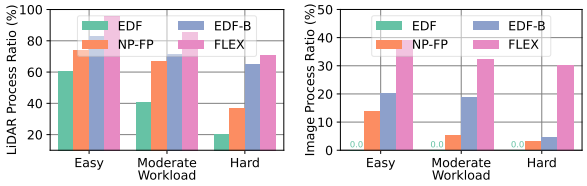
1) Baselines: All baselines are non-preemptive.

- **NP-FP [11]:** Non-preemptive fixed-priority scheduling. It is a state-of-the-art multi-task perception scheduling



(a) Detection Recall. (b) Detection Precision. (c) mAP.

Fig. 9. Scheduling algorithms comparison on detection accuracy.



(a) LiDAR Process Ratio. (b) Image Process Ratio.

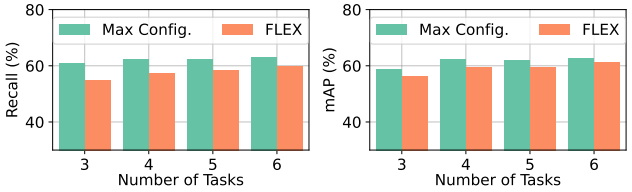
Fig. 10. Scheduling algorithms comparison on sensor data throughput.

algorithm. We use the version that allows flexible runtime priority inversion and we employ Rate Monotonic (RM) as the prioritization policy.

- **EDF:** Earliest deadline first scheduling. It always chooses the job with the earliest deadline. We use the maximum static configuration such that no deadline miss happens via offline schedulability test.
- **EDF-B:** EDF scheduling with task batching. It batches the maximum number of currently active jobs according to EDF order such that: (1) The time budget of the batch is the sum of the budgets of all jobs. (2) No deadline miss happens within the batch and no unpublished jobs may be blocked by the batch's execution.

It is worth noting that both NP-FP and EDF-B use the same elastic fusion algorithm as FLEX, while EDF uses the offline-determined static fusion configuration that ensures no deadline miss happens.

2) Scheduling Results: The experiment results on detection quality and data process ratio are shown in Figure 9 and Figure 10, respectively. We found that FLEX achieves the best in all detection quality metrics under different levels of workloads. It significantly outperforms baselines in recall, a vital metric in autonomous driving since the consequence of missing an actual object can be severe, which is typically positively related to the processed data ratio. Besides, FLEX demonstrates stability under various workloads, with only a minor accuracy degradation under the hard workload compared to the easy workload. Generally, batch scheduling frameworks can utilize the parallel computing ability of GPU to process more sensor data, which leads to improved detection quality. As depicted in Figure 10, sequential scheduling algorithms, such as NP-FP, struggle to process and fuse image features under heavy workloads, leading to a significant drop in accuracy. Besides, compared to EDF-B, FLEX triggers more batch opportunities and sets tighter deadlines for feasible batch execution with the assistance of the offline-generated schedule graph, resulting in a comparatively higher image processing ratio even under the hard workload.

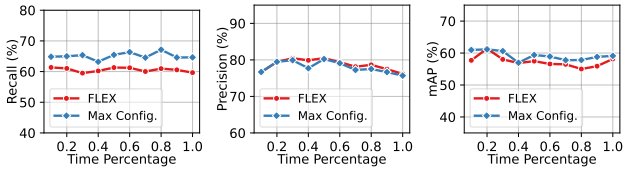


(a) Detection Recall. (b) mAP.

Fig. 11. The impact of the number of tasks on our approach.

TABLE II
BATCHED JOB PROPORTION WITH DIFFERENT NUMBER OF TASKS.

Number of Tasks	3	4	5	6
Batching Proportion	68.26%	91.51%	96.32%	94.72%
LiDAR Process Ratio	94.18%	63.50%	60.03%	74.58%
Image Process Ratio	3.87%	46.97%	51.92%	54.42%

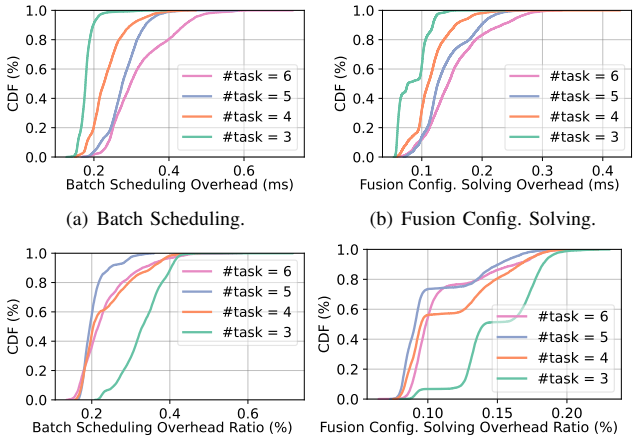


(a) Detection Recall. (b) Detection Precision. (c) mAP.

Fig. 12. Detection accuracy over time.

E. Varying Number of Views

In this section, we evaluate the impact of the task set on FLEX's performance. We repeat the experiments with 3, 4, 5, and 6 views respectively. To exclude the influence of absolute workload, we assign periods for each task set so that the task set's utilization with the minimum configuration is around 85% for all number of views. We compare the accuracy results of FLEX and the maximum configuration (5 LiDAR frames with image feature fusion, as the upper bound) in Figure 11. FLEX is generally effective with varying numbers of views, with a recall degradation from 3.1% to 5.9%, and an mAP degradation from 1.5% to 2.7%, compared to the upper bound. The degradation differs among different task sets partially because of the varying batch execution opportunities under different job publish modes. We further calculate the ratio of the batched job over the total executed jobs in Table II. FLEX generally optimizes the job batching policies to achieve higher data processing throughput and higher perception quality. When the number of tasks is relatively small, *e.g.*, 3 views, fewer task batching opportunities are identified, which leads to insufficient utilization of the available computing resources and a significant drop in image process ratio, thus the gap between FLEX and the max configuration is relatively large. We conclude that the advantage of FLEX lies in scenarios when more tasks need to be scheduled and processed creating more potential batching opportunities.



(a) Batch Scheduling. (b) Fusion Config. Solving.

(c) Batch Scheduling (Ratio). (d) Fusion Config. Solving (Ratio).

Fig. 13. Scheduling overhead breakdown in FLEX.

F. Temporal Detection Quality Consistency

One concern on FLEX is whether the "aggressive" batching in the scheduling stage of early jobs can occupy the execution time and sacrifice the detection quality of later jobs. We further investigate the changes in detection accuracy over time under the FLEX scheduling. We divide each scene into ten equal time intervals and record the accuracy for each interval. We compare the results with the maximum configuration (5 LiDAR frames with image feature fusion) as shown in Figure 12. FLEX maintains high detection quality throughout entire scenes, with a relatively small gap compared to the maximum configuration. FLEX determines the time budget for the current batch while ensuring sequential schedulability for future jobs, which preserves sufficient batch execution time for both current and future jobs. This leads to a balanced temporal load and resource allocation, thereby guaranteeing the stability of detection quality over time.

G. Scheduling Overhead

Finally, we report the breakdown scheduling overhead of FLEX. The results are shown in Figure 13 as cumulative distribution functions (CDF). We evaluate the overhead with different numbers of tasks and report both the absolute scheduling latency and its relative ratio over its batch execution time. The scheduling overhead consists of two parts: generating feasible batch candidates and deciding the fusion configurations for batch execution. Both are lightweight compared to the job's execution latency in most cases, and the former is minimized in our approach with the help of the offline-generated look-up table. Our scheduling process runs on a CPU and doesn't compete for GPU resources with the detection model, though the scheduling overhead is counted into the overall end-to-end latency, and it can be simply subtracted from the time budget assigned to the batch execution.

VII. Discussion and Limitations

Scalability of HSD construction. We adopt node merging in the HSD construction process to partially alleviate state explosion. Two factors affect the scalability of HSD construction: First, the camera count, which is limited and 6 cameras already match existing AV systems. Second is the hyper-period length, which is determined by the task period composition. Our experiments include 5-40s hyper-periods with 80-900 jobs where the HSD construction takes 40s-15min on a desktop, which is acceptable. The real-world task sets in industrial practice typically contain a few hundred to a few thousand jobs in a hyper-period [54], [55]. The hyper-period may be excessively long, e.g., in non-harmonic [56] period compositions, in which case more empirical optimization techniques need to be explored.

Preemption of DNN Inference. We use the non-preemptive task model because commodity GPUs lack efficient preemptive scheduling support [25], [57]. Mainstream deep-learning frameworks [58], [59] usually provide only limited preemption control via the coarse-grained GPU stream priority. Though existing works explore software-based methods, e.g., chunk-wise DNN [60], or hardware enhancement, e.g., lightweight context-switch [61], [62] to promote GPU preemption, their approaches are limited (e.g., the preemption granularity of chunk-wise methods is determined by the execution time of the DNN chunk) and more hardware supports are required. To extend FLEX to preemptive cases while still fully exploiting the massive parallelism capability of the GPU, one of the future directions is to design continuous batching techniques [63] so that a running batch may dynamically integrate parts of new DNN jobs for more fine-grained parallelism.

VIII. Conclusion

We presented a novel real-time scheduling framework, FLEX, for multi-modal multi-view perception systems on resource-constrained embedded platforms equipped with an onboard GPU. It effectively combines an elastic multi-modal fusion strategy with an adaptive batch scheduling algorithm, in a context-aware scheduling principle, to appropriately allocate the limited computing resources to the critical spatial views with more objects. Extensive evaluations are performed with a large-scale real-world autonomous driving dataset, including 85 driving scenarios, on the Jetson Orin platform, to demonstrate the effectiveness of FLEX in achieving superior resource allocation between different views and modalities that lead to consistently better detection quality and data processing throughput than the baselines.

Acknowledgement

This work was sponsored in part by the National Key R&D Program of China (No. 2022ZD0119100), in part by China NSF grant No. 62472278, 62025204, 62432007, 62332014, and 62332013, in part by Alibaba Group through Alibaba Innovation Research Program, and in part by Tencent Rhino Bird Key Research Project. This work was partially supported by SJTU Kunpeng&Ascend Center of Excellence. The opinions, findings, conclusions, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies or the government.

REFERENCES

[1] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Gläser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1341–1360, 2021.

[2] L. Wang, X. Zhang, Z. Song, J. Bi, G. Zhang, H. Wei, L. Tang, L. Yang, J. Li, C. Jia, and L. Zhao, "Multi-modal 3d object detection in autonomous driving: A survey and taxonomy," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 7, pp. 3781–3798, 2023.

[3] R. Qian, X. Lai, and X. Li, "3d object detection for autonomous driving: A survey," *Pattern Recognition*, vol. 130, p. 108796, 2022.

[4] H. Li, C. Sima, J. Dai, W. Wang, L. Lu, H. Wang, J. Zeng, Z. Li, J. Yang, H. Deng, H. Tian, E. Xie, J. Xie, L. Chen, T. Li, Y. Li, Y. Gao, X. Jia, S. Liu, J. Shi, D. Lin, and Y. Qiao, "Delving into the devils of bird's-eye-view perception: A review, evaluation and recipe," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–20, 2023.

[5] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang, L. Lu, X. Jia, Q. Liu, J. Dai, Y. Qiao, and H. Li, "Planning-oriented autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[6] Y. Man, L.-Y. Gui, and Y.-X. Wang, "Bev-guided multi-modality fusion for driving perception," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 21 960–21 969.

[7] Z. Song, L. Yang, S. Xu, L. Liu, D. Xu, C. Jia, F. Jia, and L. Wang, "Graphbev: Towards robust bev feature alignment for multi-modal 3d object detection," *arXiv preprint arXiv:2403.11848*, 2024.

[8] S. Liu, X. Fu, M. Wigness, P. David, S. Yao, L. Sha, and T. Abdelzaher, "Self-cueing real-time attention scheduling in criticality-aware visual machine perception," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022, pp. 173–186.

[9] W. Kang, S. Chung, J. Y. Kim, Y. Lee, K. Lee, J. Lee, K. G. Shin, and H. S. Chwa, "Dnn-sam: Split-and-merge dnn execution for real-time object detection," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022, pp. 160–172.

[10] S. Liu, S. Yao, X. Fu, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha, and T. Abdelzaher, "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 319–332.

[11] D. Kang, S. Lee, H. S. Chwa, S.-H. Bae, C. M. Kang, J. Lee, and H. Baek, "Rt-mot: Confidence-aware real-time scheduling framework for multi-object tracking tasks," in *2022 IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 318–330.

[12] X. Hou, P. Tang, C. Li, J. Liu, C. Xu, K.-T. Cheng, and M. Guo, "Smg: A system-level modality gating facility for fast and energy-efficient multimodal computing," in *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023, pp. 291–303.

[13] X. Hou, J. Liu, X. Tang, C. Li, K.-T. Cheng, L. Li, and M. Guo, "Mmexit: Enabling fast and efficient multi-modal dnn inference with adaptive network exits," in *European Conference on Parallel Processing*. Springer, 2023, pp. 426–440.

[14] Z. Liu, H. Tang, A. Amini, X. Yang, H. Mao, D. Rus, and S. Han, "Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[15] A. Haidar, T. Dong, P. Luszczek, S. Tomov, and J. Dongarra, "Batched matrix computations on hardware accelerators based on gpus," *The International Journal of High Performance Computing Applications*, vol. 29, no. 2, pp. 193–208, 2015.

[16] S. Lee and S. Nirjon, "Subflow: A dynamic induced-subgraph strategy toward real-time dnn inference and training," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 15–29.

[17] S. Heo, S. Cho, Y. Kim, and H. Kim, "Real-time object detection system with multi-path neural networks," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 174–187.

[18] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher, "Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, 2018, pp. 278–291.

[19] S. Bateni and C. Liu, "Apnet: Approximation-aware real-time neural network," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, 2018, pp. 67–79.

[20] W. Jang, H. Jeong, K. Kang, N. Dutt, and J.-C. Kim, "R-tod: Real-time object detector with minimized end-to-end delay for autonomous driving," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 191–204.

[21] M. Ji, S. Yi, C. Koo, S. Ahn, D. Seo, N. Dutt, and J.-C. Kim, "Demand layering for real-time dnn inference with minimized memory usage," in *2022 IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 291–304.

[22] T. Kannan and H. Hoffmann, "Budget rnns: Multi-capacity neural networks to improve in-sensor inference under energy budgets," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 143–156.

[23] A. Soyyigit, S. Yao, and H. Yun, "Anytime-lidar: Deadline-aware 3d object detection," in *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2022, pp. 31–40.

[24] Y. Hu, I. Gokarn, S. Liu, A. Misra, and T. Abdelzaher, "Algorithms for canvas-based attention scheduling with resizing," in *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024, pp. 348–359.

[25] W. Kang, K. Lee, J. Lee, I. Shin, and H. S. Chwa, "Lalarand: Flexible layer-by-layer cpu/gpu scheduling for real-time dnn tasks," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 329–341.

[26] Y. Xiang and H. Kim, "Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, 2019, pp. 392–405.

[27] H. Zhou, S. Bateni, and C. Liu, "S^3dnn: Supervised streaming and scheduling for gpu-accelerated real-time dnn workloads," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018, pp. 190–201.

[28] Z. Li, T. Ren, X. He, and C. Liu, "Red: A systematic real-time scheduling approach for robotic environmental dynamics," in *2023 IEEE Real-Time Systems Symposium (RTSS)*, 2023, pp. 210–223.

[29] L. Liu, Z. Dong, Y. Wang, and W. Shi, "Prophet: Realizing a predictable real-time perception pipeline for autonomous vehicles," in *2022 IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 305–317.

[30] Z. Yang, K. Nahrstedt, H. Guo, and Q. Zhou, "Deeprt: A soft real time scheduler for computer vision applications on the edge," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2021, pp. 271–284.

[31] S. Liu, S. Yao, X. Fu, H. Shao, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha, and T. Abdelzaher, "Real-time task scheduling for machine perception in intelligent cyber-physical systems," *IEEE Transactions on Computers*, vol. 71, no. 8, pp. 1770–1783, 2021.

[32] J. Arevalo, T. Solorio, M. Montes-y Gómez, and F. A. González, "Gated multimodal units for information fusion," *arXiv preprint arXiv:1702.01992*, 2017.

[33] R. Panda, C.-F. R. Chen, Q. Fan, X. Sun, K. Saenko, A. Oliva, and R. Feris, "Adamml: Adaptive multi-modal learning for efficient video recognition," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 7576–7585.

[34] S. Vora, A. H. Lang, B. Helou, and O. Beijbom, "Pointpainting: Sequential fusion for 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[35] F. Wulff, B. Schäufele, O. Sawade, D. Becker, B. Henke, and I. Radusch, "Early fusion of camera and lidar for robust road detection based on u-net fcn," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1426–1431.

[36] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[37] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–8.

[38] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun, "Multi-task multi-sensor fusion for 3d object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 7345–7353.

[39] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep continuous fusion for multi-sensor 3d object detection," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 641–656.

[40] S. Pang, D. Morris, and H. Radha, "Clocs: Camera-lidar object candidates fusion for 3d object detection," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 10 386–10 393.

[41] L. Agostinho, D. Pereira, A. Hiolle, and A. Pinto, "Tefu-net: A time-aware late fusion architecture for robust multi-modal ego-motion estimation," *Robotics and Autonomous Systems*, p. 104700, 2024.

[42] J. W. Liu, K.-J. Lin, and S. Natarajan, "Scheduling real-time, periodic jobs using imprecise results," Tech. Rep., 1987.

[43] J. W.-S. Liu, K.-J. Lin, W. K. Shih, A. C.-s. Yu, J.-Y. Chung, and W. Zhao, *Algorithms for scheduling imprecise computations*. Springer, 1991.

[44] J. W.-S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise computations," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 83–94, 1994.

[45] S. Yao, Y. Hao, Y. Zhao, H. Shao, D. Liu, S. Liu, T. Wang, J. Li, and T. Abdelzaher, "Scheduling real-time deep learning services as imprecise computations," in *2020 IEEE 26th international conference on embedded and real-time computing systems and applications (RTCSA)*. IEEE, 2020, pp. 1–10.

[46] J. Sun, T. Wang, Y. Li, N. Guan, Z. Guo, and G. Tan, "Seam: An optimal message synchronizer in ros with well-bounded time disparity," in *2023 IEEE Real-Time Systems Symposium (RTSS)*, 2023, pp. 172–184.

[47] X. Chen, S. Shi, B. Zhu, K. C. Cheung, H. Xu, and H. Li, "Mppnet: Multi-frame feature intertwining with proxy points for 3d temporal object detection," in *European Conference on Computer Vision*. Springer, 2022, pp. 680–697.

[48] D. P. Bertsekas, *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[49] C. Ekelin, "Clairvoyant non-preemptive edf scheduling," in *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, 2006, pp. 7 pp.–32.

[50] K. Jeffay, D. Stanat, and C. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *[1991] Proceedings Twelfth Real-Time Systems Symposium*, 1991, pp. 129–139.

[51] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *CVPR*, 2020.

[52] M. Contributors, "MMDetection3D: OpenMMLab next-generation platform for general 3D object detection," https://github.com/open-mmlab/mmdetection3d, 2020.

[53] NVIDIA, "Lidar_AI_Solution," https://github.com/NVIDIA-AI-IOT/Lidar_AI_Solution, 2024.

[54] S. Anssi, S. Kuntz, S. Gérard, and F. Terrier, "On the gap between schedulability tests and an automotive task model," *Journal of Systems Architecture*, vol. 59, no. 6, pp. 341–350, 2013.

[55] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, vol. 130, 2015.

[56] Y. Cai and M. Kong, "Nonpreemptive scheduling of periodic tasks in uni-and multiprocessor systems," *Algorithmica*, vol. 15, pp. 572–599, 1996.

[57] M. Han, H. Zhang, R. Chen, and H. Chen, "Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 539–558. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/han

[58] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[59] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "{TensorFlow}: a system for {Large-Scale} machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[60] L. Han, Z. Zhou, and Z. Li, "Pantheon: Preemptible multi-dnn inference on mobile edge gpus," ser. MOBISYS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 465478. [Online]. Available: https://doi.org/10.1145/3643832.3661878

[61] J. J. K. Park, Y. Park, and S. Mahlke, "Chimera: Collaborative preemption for multitasking on a shared gpu," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1, pp. 593–606, 2015.

[62] I. Tanasic, I. Gelado, J. Cabezas, A. Ramirez, N. Navarro, and M. Valero, "Enabling preemptive multiprogramming on gpus," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 193–204, 2014.

[63] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for Transformer-Based generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 521–538. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/yu