Jigsaw: Taming BEV-centric Perception on Dual-SoC for Autonomous Driving

Lingyu Sun[1], Chao Li[1], Xiaofeng Hou[1], Tianhao Huang[1], Cheng Xu[1],
Xinkai Wang[1], Guangjun Bao[2], Bingchuan Sun[2], Shibo Rui[2], Minyi Guo[1]
[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[2]Vehicle Computing, Lenovo Group, Beijing, China
sunlingyu@sjtu.edu.cn, {lichao, hou-xf}@cs.sjtu.edu.cn, {hth2003, jerryxu, unbreakablewxk}@sjtu.edu.cn,
{jbao, sunbc1, ruisb1}@lenovo.com, guo-my@cs.sjtu.edu.cn

Abstract*—Real-time perception is important for autonomous driving. We observe an emerging trend using one large and critical fusion-based Bird's-Eye-View (BEV) Deep Neural Network (DNN) model to perform core perception tasks. It collaborates with a few auxiliary Perspective-View (PV) models, forming a *BEV-centric* paradigm. Organizing the BEV and PV models respecting their distinct real-time requirements becomes challenging, especially on the state-of-the-practice GPU-integrated dual System-on-Chip (SoC) platform. It remains unclear how to appropriately allocate the separated GPU resource to BEV and PV models, satisfying their distinct real-time requirements with latency predictability. *No public solution has been proposed for this emerging software-hardware combination.

This paper explores parallelism and a timeslot-filling mechanism to organize tasks. We propose *Jigsaw*, a specialized execution timeline management framework for BEV-centric perception on dual-SoC. First, it exploits *component parallelism* to carefully place BEV model components and reduce BEV model latency. Second, we recognize two types of idle GPU timeslots left by a parallelized BEV model. The stable timeslot can offer hard real-time guarantee for PV models, while the unstable timeslot could only provide soft real-time capability. Therefore, *Jigsaw* schedules PV models by *timeslot filling* to ensure latency predictability of BEV model and deadline satisfaction of PV models. The framework is implemented in compliance with the practical computing stack in modern autonomous vehicles. It is evaluated on a dual-SoC prototype connected via a PCIe bus. Results show that it achieves 1.52-1.63× speedup for the BEV model compared to no parallelism. It also ensures deadline satisfaction for PV models without interference in BEV model latency predictability.

I. Introduction

Modern automotive vehicles are developing towards autonomy [1]. An increasing number of them are equipped with Advanced Driver Assistance Systems (ADAS) achieving near self-driving [2]. In the automotive industry, the corresponding computing stack is integrated into the *ADAS domain*.

The ADAS domain works in real time. As shown in Fig. 1, its workload is organized as a periodic three-stage pipeline including sensing, perception, and planning [3]–[5]. Perception is recognized as the performance bottleneck, accounting for more than 94% computation due to the excessive usage of Deep Neural Networks (DNNs) [3]. Underneath, the state-of-the-practice platform, or *ADAS domain controller*, is backed by System-on-Chips (SoCs) due to Space, Weight, and Power (SWaP) constraints. Currently, the most popular setup consists of two homogeneous inter-connected SoCs for fault tolerance

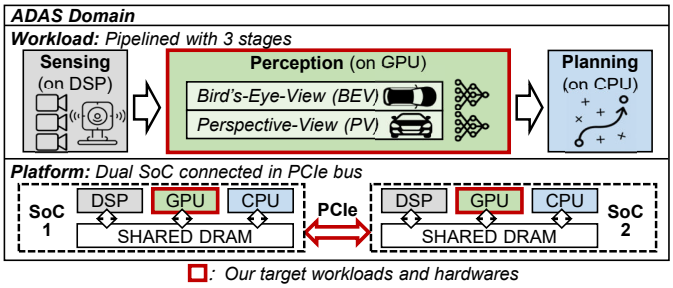

Fig. 1: Software-hardware architecture of ADAS domain.

under emergency while achieving higher total performance in normal times [6], [7]. The SoCs can achieve high-speed data transfer via PCIe buses [8], [9]. Each SoC is equipped with an integrated GPU (i.e. no private GPU memory) for DNN acceleration [10], [11].

Traditionally, the perception stage employs multiple independent GPU-accelerated algorithms to process each sensor and each task [5], [12], shown in Fig. 2(a). Very recently, things have changed as algorithm researchers turn to a centralized perception workflow for better sensor synergy, shown in Fig. 2(b). It primarily relies on a single fusion-based DNN model which is able to process multi-modal sensor inputs jointly and generate multi-task results [13]–[19]. This model works in Bird's Eye View (BEV). It is able to handle the most critical tasks for collision avoidance. Only a few auxiliary tasks still need independent image DNNs in Perspective View (PV). We summarize this workflow as *BEV-centric* perception. On a dual-SoC platform, simply distributing the BEV and PV models to distinct SoCs is suboptimal. The critical BEV model could not enjoy parallel speedup, while PV models may not fully occupy a GPU when their arrival is infrequent.

Therefore, it becomes an important problem to properly organize the centralized workflow on the distributed platform, satisfying the respective real-time requirements of BEV and PV models. From our investigation of public information and open-source frameworks (see Sec. II), we target the following setup: The BEV model should run consecutively as fast as possible with stable latency for safety and predictability [12]. The PV models should run periodically under relatively loose

but diverse deadlines. In this paper's remaining parts, *BEV/PV model* and *BEV/PV task* are used interchangeably.

Although many prior works focus on ADAS perception workload management, they are not quite suitable for the combination of the emerging BEV-centric paradigm and distributed dual-SoC. Some works try to coordinate concurrent multi-DNN execution on a single server-class GPU [20]–[22]. However, concurrency brings little benefit but causes interference on a SoC-integrated GPU since its hardware resource (e.g. streaming processors) is much less than a server-class one. Other works propose methods applicable to organizing multi-DNNs on a distributed platform through run-time decision-making, including GPU frequency adjustment [23], [24] or DNN early exit [4], [12]. These approaches require complicated implementation. Their designs do not distinguish the BEV and PV models and hence fail to prioritize latency reduction of the critical BEV model.

In this paper, we propose *Jigsaw*, an execution timeline management framework specialized for BEV-centric perception on a dual-SoC platform. It prioritizes parallel speedup of the critical BEV task and executes the auxiliary PV tasks exploiting the remaining fragmented GPU time. We first observe that the fusion-based BEV model has multiple components [13]–[15]. We discover that placing independent components on distinct SoCs can be quite beneficial for latency reduction. The execution time of the middling unparallelizable fuser component is small, while the execution time of parallelizable backbone and head components is comparable. The intermediate communication overhead is not large. Therefore, *Jigsaw* first exploits *component parallelism* and decides the component placement of the BEV model during offline analysis. It carefully places the fuser to overlap communication overhead with computation and redistributes the input batch of a long image backbone for better workload balancing.

After BEV model parallelization, we identify two types of idle GPU timeslots left that can be used to accommodate PV models. *Stable timeslots* reside on the SoC not executing the BEV model's fuser component. It has a fixed length and appears in the middle of every BEV model execution iteration. *Unstable timeslots* appear on SoCs executing a special kind of BEV model's backbone component. It has a variable length in each BEV model execution iteration. In detail, it results from the usage of voxel-based point cloud backbones whose execution time is intrinsically variable due to sparsity [25], [26]. We further show that the native GPU scheduler fails to utilize these timeslots without affecting the BEV model latency. Performance interference happens due to multi-task contention [27], [28] and a lack of sound preemption support [29], [30]. Therefore, *Jigsaw* schedules PV tasks by *timeslot filling* and bypasses the GPU native scheduler. Each GPU is treated as a preemptive uniprocessor. Hard real-time PV tasks are scheduled to stable timeslots, while soft real-time ones go to unstable timeslots. It first splits up PV models into pieces before scheduling for controlled preemption. The piece size is smaller than the timeslot size for better utilization and schedulability. It then conducts formal schedulability analysis to ensure deadline satisfaction. During runtime, the *Jigsaw* framework integrates a lightweight scheduler for each timeslot. It receives PV task requests periodically and dispatches the model pieces without affecting the BEV model's latency.

Our paper makes the following contributions:

- We gain an in-depth understanding of the BEV model's latency characteristics and discover a speedup opportunity by exploiting component parallelism. We further reveal the existence of two types of idle timeslots after BEV model parallelization.
- We propose a specialized timeline management framework, *Jigsaw*. For the BEV model, *Jigsaw* decides the parallelism details to achieve minimal latency. For the PV models, *Jigsaw* schedules them into the timeslots in units of model pieces and conducts schedulability analysis to ensure no deadline violation or performance interference.
- We provide an easy-to-integrate and ROS 2 compatable implementation of *Jigsaw*. We evaluate *Jigsaw* using two NVIDIA AGX Orin SoCs connected via a PCIe buse. Results show 1.52-1.63$\times$ parallel speedup for four popular BEV model variants compared to no parallelism. When dispatching PV tasks into timeslots, it can ensure stable BEV model latency and always satisfy PV model deadlines. For PV models with a total execution time of less than 100 ms, the stable timeslot can accommodate them at over 2-3 FPS. The unstable timeslot can hold them at over 1 FPS with a 90% deadline satisfaction rate.

II. Background

ADAS perception workload. As shown in Fig. 2(a), the perception input mainly comes from point clouds and images [5]. LiDAR, radar, and millimeter-wave sensors generate point clouds, while camera and infrared sensors produce images. LiDAR and cameras are the most commonly-used sensors. Traditionally, the perception stage processes each input and task independently. The outputs of algorithms using different inputs for the same task are fused later during planning (i.e. late fusion [31]). This workflow suits the distributed dual-SoC platform well. Independent algorithms can be distributed to distinct SoCs for parallel speedup. However, its perception accuracy is degraded due to late fusion. Redundant computation is introduced when processing the same sensor input for different tasks.

BEV-centric perception workflow has recently gained popularity, as shown in Fig. 2(b). A fusion-based BEV model integrates multi-modal sensor inputs in BEV space [13]–[19]. The fused BEV feature divides the vehicle surroundings into small grids and describes the attributes of each grid. Therefore, it can handle several most critical tasks for collision avoidance, including object detection, drivable area detection, localization, etc [14], [15], [32]. The BEV model also manifests itself in its uniform and modular design [14]–[19]. Its point cloud backbones (PB), image backbones (IB), fuser (FU), and task heads (TH) can all reuse existing solutions. Different component choices have distinct tradeoffs in execution latency and overall accuracy. For example, its camera IB can be either

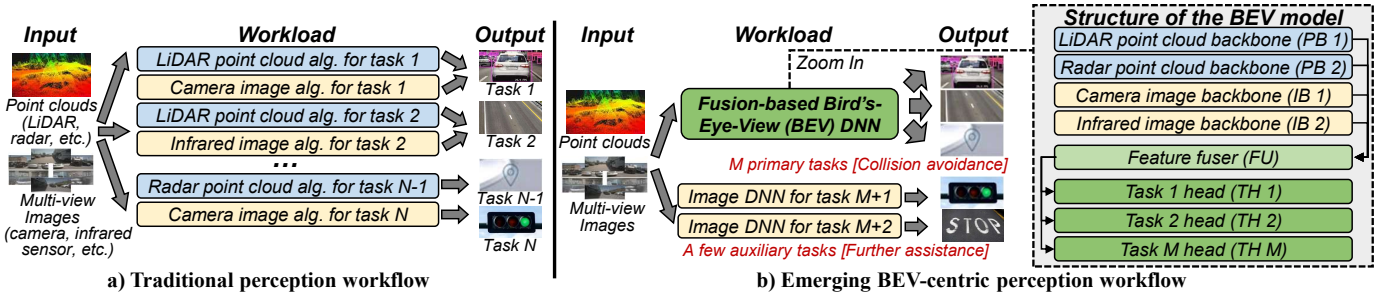

Fig. 2: Workflow comparison of the traditional perception and the emerging BEV-centric perception.

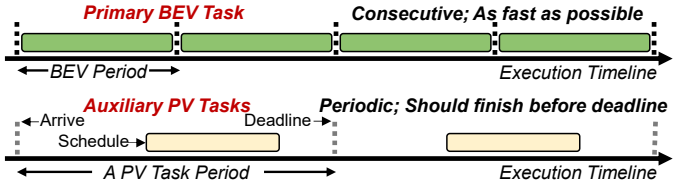

Fig. 3: Task execution pattern of BEV-centric perception.

a convolutional Resnet or a vision transformer. Only a few auxiliary tasks still need participation from independent image DNNs [13]. These tasks are not suitable in BEV and require fine-grained semantic information in PV (e.g., traffic light detection, road mark recognition, etc.).

ADAS domain platform. Today's vehicle manufacturers commonly employ homogeneous dual-SoC as the ADAS domain controller. For example, many modern vehicles are equipped with NVIDIA dual Drive Orin SoC solution [33]. Tesla's vehicles use dual FSD SoC solution [7]. Few manufacturers integrate up to 3-4 SoCs as the ADAS domain controller [34]. The most popular NVIDIA Drive Orin SoC is equipped with an integrated GPU, while non-NVIDIA platforms are equipped with ASIC AI accelerators. The SoCs commonly use a shared main memory between CPU, GPU, and DSP. They have a PCIe bus connection for fast data transfer and another Ethernet connection for control-flow and low-speed data exchange. This paper mainly uses the GPU-integrated NVIDIA Orin SoC as the platform since it is the only publicly available ADAS SoC. Our proposed *Jigsaw* framework is applicable to non-GPU platforms and can be extended to multi-SoC platforms, which is discussed in Sec. VII.

Task execution pattern. Generally, ADAS domain tasks run periodically. According to prior works, the execution frequency of the ADAS task pipeline should be higher than 10 FPS (100 ms period) [3], [5], [12]. Higher execution frequency is also preferred for faster reaction and smoother experience as long as there is new data from the camera and LiDAR sensors (typically 30-60 FPS, 16-33 ms period). However, not all perception tasks are required to execute so frequently. We have examined two popular open-sourced ADAS software suites and find the execution frequency of traffic light detection task is set at 0.5 and 8 FPS in Autoware [35] and Apollo [36] respectively. Therefore, in this paper, we summarize the task execution pattern in Fig. 3. The BEV task for collision avoidance should run consecutively as fast as possible since the execution time of the BEV model is generally longer than the sensor period (Section III-A). The latency of the BEV task, or *BEV Period*, should be stable. AuxiliaryPV tasks should run periodically under deadlines equal to the arrival periods. Different PV tasks may have diverse periods. We currently do not factor in aperiodic or sporadic tasks.

III. Motivation

A. Speedup Opportunity for BEV Model

To examine how much parallel speedup opportunity exists for the BEV model, we examine the component execution time and communication overhead of four popular BEV model variants from small to large in Tab. I. Each of them has a LiDAR PB, a camera IB, an object detection task head (DH), and a scene segmentation task head (SH). This is the most common BEV model structure at present while having more backbones and heads is also possible. We measure the 99th tail execution time of each component on the popular Nuscenes [37] dataset containing 6019 samples. We test on NVIDIA AGX Orin SoC, whose automotive-grade version is widely used for ADAS.

As shown in Fig. 4(a), the execution time of selected BEV models lies in a reasonable range from around 35 to 120 ms. The fuser component only accounts for a tiny portion of the total execution time. Therefore, the BEV model is parallelizable to a large extent owing to its intrinsic structure. We can observe execution time imbalance in the parallelizable components, which could attenuate latency reduction (e.g., IB is much longer than PB for BEV models 1 and 2). However, we could redistribute the input batch of a long IB to balance the backbones (Sec. IV-B).

Another decisive precondition of the applicability of parallelism lies in the speed of intermediate feature data transfer between the two SoCs. Therefore, we measure the transfer time of the LiDAR feature, camera feature, and fused BEV features over PCIe 3.0×8 (up to 64 Gb/s). Since AGX Orin is not equipped with the necessary onboard PCIe switch chip as its automotive-grade version, we attach additional high-speed network cards with Remote Direct Memory Access (RDMA) support on its PCIe slot, providing up to 50 Gb/s bandwidth.

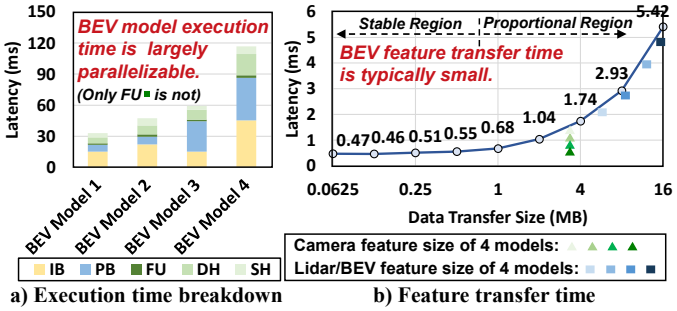

Fig. 4: Execution time and intermediate data transfer time of the selected BEV models.

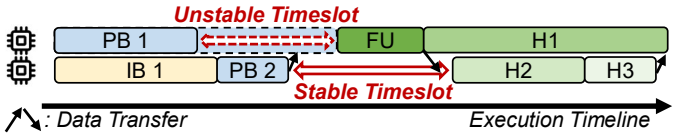

Fig. 5: A sketch of available timeslots after achieving parallel speedup for BEV model.

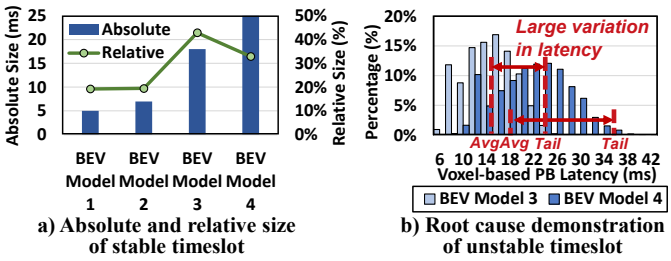

Fig. 6: Quantitative analysis on stable and unstable timeslots.

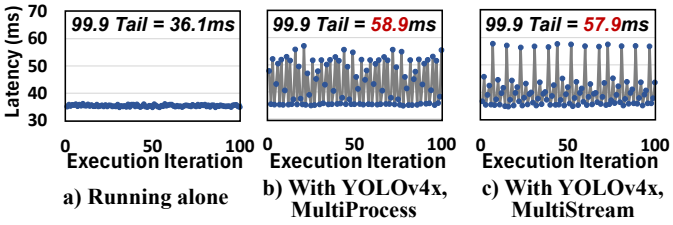

Fig. 7: Execution latency of BEV model under different co-run setups using Nuscenes dataset.

As shown in Fig. 4(b), the transfer latency increases almost linearly at nearly 24 Gb/s after data size exceeds 1MB. Peak bandwidth is not reached since the transfer size is small and not batched. However, this is fast enough to transfer the intermediate features at around 1-5 ms. Such transfer latency can be largely overlapped with computation, bringing minimal influence on parallel speedup (Section IV-B). The task results are less than 1 MB and have negligible overhead (not shown). It is worth mentioning that the intermediate data and task results from the GPU can be directly sent by the CPU since the SoC follows a shared memory architecture.

In summary, parallel speedup opportunity exists for the BEV model exploiting its intrinsic structure. Its execution time is largely parallelizable, while the intermediate data transfer time is typically small and can be overlapped.

B. Available Timeslot for PV Models

After achieving parallel speedup for the BEV model, we identify two types of idle GPU timeslots left to accommodate PV models. Fig. 5 shows a sketch of the timeslots after parallelizing a BEV model with three backbones and heads.

Stable timeslot resides on the SoC not executing the FU component. Its size remains stable in each BEV period. It comes from two aspects: the unparallelizable FU, and the imbalanced backbones and heads. Fig. 6(a) shows the size of stable timeslot for the four selected BEV models on dual-SoC after applying details discussed in Sec. IV-B. Its absolute size (in milliseconds) ranges from 5 to 25 ms for different BEV models, and the relative size (in percentage, divided by BEV period length) ranges from 20% to 40%. Properly utilizing it could provide a hard real-time guarantee for PV tasks.

Unstable timeslot stems from the latency variable nature of PB, while other BEV model components generally show minimal latency variation. There are mainly three types of point cloud processing algorithms. *Point-based* ones, including PointNet++ [38], process raw points offering highest accuracy. However, it is currently prohibitively compute-heavy for edge SoCs [39]. *Voxel-based* ones, including VoxelNet [25], voxelize the raw points and use 3D sparse convolution (SpConv3D) to exploit its intrinsic sparsity for better compute efficiency. Its execution time depends on input point cloud distribution. *Pillar-based* ones, including PointPillars [40], pillarize the raw points and process them using standard convolution. Its execution time is stable and the shortest but offers the lowest accuracy. Current BEV models can be integrated with either voxel-based or pillar-based point cloud backbones [14], [15]. In Tab. I, the BEV models 1 and 2 use pillar-based PB, while models 3 and 4 use voxel-based PB. Fig. 5 demonstrates a schematic diagram of a BEV model with two PBs. PB1 is voxel-based and PB2 is pillar-based.

Fig. 6(b) shows the execution time distribution of the voxel-based PB used in BEV models 3 and 4 with different input voxel resolutions. We can observe a large gap between its average and tail latency. Consequently, to ensure a stable BEV period, the execution of voxel-based PB should be padded to its tail latency, leaving an unstable timeslot in each BEV period. Unstable timeslots can be utilized for soft real-time PV tasks that expect deadline satisfaction under a certain percentile (e.g., 90% or 99%).

In summary, there are two types of GPU timeslot left (stable and unstable) after deploying the BEV model in parallel with predictable latency. PV models can be accommodated in these timeslots without influencing BEV model latency.

C. Uncoordinated Concurrent Execution

In this subsection, we check the effectiveness of the native scheduling mechanisms provided by commercial off-the-shelf GPUs in terms of properly utilizing available timeslots to

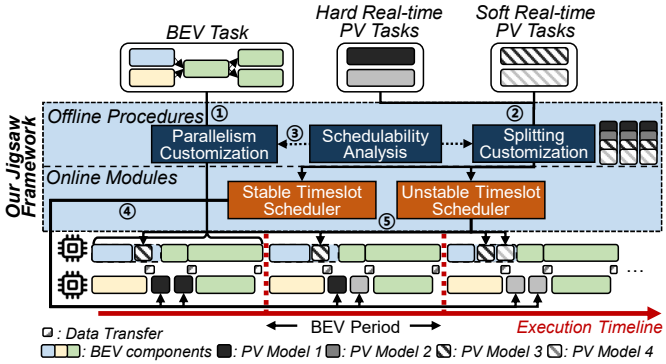

Fig. 8: Overview of *Jigsaw* framework.

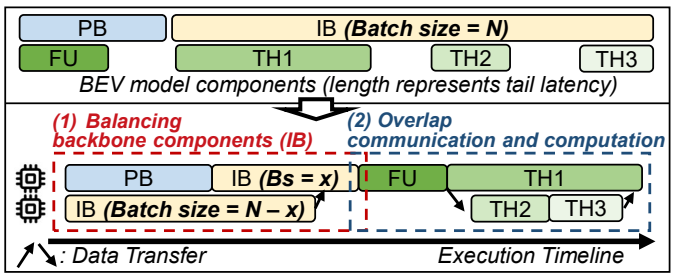

Fig. 9: An illustration of the two customization steps involved to achieve better parallel speedup for the BEV model.

accommodate large PV tasks without affecting the latency predictability of the BEV model.

As opposed to CPUs, the multi-tasking support of GPUs is quite opaque and deficient. Modern NVIDIA GPUs support multi-tasking through a black-box scheduler in the device driver under two cases. **1)** *MultiProcess case:* When GPU tasks are submitted by CPU from different operating system processes, the scheduler would execute their *kernels* (the scheduling unit of GPU tasks) concurrently by time slicing. Every kernel from each task monopolizes the GPU and is scheduled in a *nondeterministic* manner [27], [28]. Although being intrinsically non-predictable, MultiProcess is the most common practice to organize GPU multi-tasks due to its simplicity. **2)** *MultiStream case:* When GPU tasks are submitted by CPU from different threads in the same process, each thread can additionally specify a *GPU stream* to use. Kernels from different streams *may* execute simultaneously if there is enough GPU hardware resource. Although each stream has a scheduling priority, kernels in high-priority stream *may* be preferred during scheduling with no strict guarantee according to documentation [41]. MultiStream is less used due to extra programmer effort to organize all tasks in a single process. The integrated GPU does not support NVIDIA MPS [42].

To quantitatively show the deficiency, we build an illustrative setup where a single PV model, YOLOv4x [43], co-runs with parallelized BEV model 2 on SoC 2 having a stable timeslot. The execution time of YOLOv4x is longer than the stable timeslot size. We hence set the value of its arrival period such that the total available timeslot length within the period is just enough to hold its execution time. A good mechanism should properly schedule YOLOv4x into the timeslots without affecting BEV model latency. We plot the end-to-end execution time of the BEV model in three scenarios: running alone, co-run under MultiProcess, and MultiStream. In MultiStream setup, the stream priority for BEV model components running on SoC 2 is set to be higher than YOLOv4x. Results of BEV model latency during the first 100 iterations are shown in Fig. 7. The BEV model's tail latency increases greatly, canceling out the parallelism speedup. Even if stream priority is set in MultiStream setup, it does not help with tail latency.

In summary, BEV and PV multi-tasking using native GPU scheduler increases the BEV model's tail latency. We have to build our own scheduling solution to properly accommodate large PV tasks in the timeslots left by the BEV model.

IV. Method

A. Jigsaw Framework Overview

We propose *Jigsaw*, a specialized execution timeline management framework, to achieve minimal and stable latency for the BEV task while accommodating PV tasks within their deadlines. *Jigsaw* is intended for ADAS solution providers whose responsibility covers deploying a given set of BEV-centric perception tasks on a dual-SoC platform under real-time requirements. It does not affect the accuracy of all DNNs.

Fig. 8 shows the architecture overview of *Jigsaw* framework. During the offline phase before deployment, it performs three preparatory procedures. *Parallelism Customization* decides details on how to execute the BEV model in parallel, achieving maximum latency reduction (①). *Splitting Customization* splits large PV models into pieces smaller than timeslot size (②). Thereafter, *Schedulability Analysis* gives the feasible lower bound of arrival periods for PV tasks such that they could be accommodated in timeslots (③). During the online phase after deployment, two scheduler modules integrated into the ADAS domain runtime receive execution requests periodically from PV tasks. *Stable Timeslot Scheduler* handles hard real-time tasks (④), while *Unstable Timeslot Scheduler* is responsible for soft real-time tasks (⑤). *Jigsaw* achieves a well-organized execution timeline, which is shown in the bottom part of Fig. 8.

B. Parallelism Customization for BEV Model

The *Parallelism Customization* generates a component placement scheme for a given BEV model achieving maximal latency reduction. This customization include two steps.

Jigsaw first redistributes the input batch of a longer IB. Each IB essentially processes a batch of homogeneous sensors (cameras) around the vehicle to obtain a panoptic result. We can hence redistribute them as illustrated in *step (1)* of Fig. 9. This could be particularly useful for BEV models using pillar-based PB (e.g., BEV models 1 and 2) whose execution time is usually much shorter than IB (see Fig. 4(a)). A long PB cannot be redistributed since point clouds from multiple radar/LiDAR sensors are commonly stitched during the sensing stage.

Algorithm 1 PV Model Automatic Split

Input: G ▷ Graph representation of the PV model
Input: t_{unit} ▷ Split piece size in milliseconds
Input: t_{delta} ▷ Split error bound in milliseconds
Output: $\mathbb{G}[]$ ▷ Graph representations of the split pieces

1: $G_{cand} \leftarrow \emptyset$; ▷ First piece candidate
2: **while** $G \neq \emptyset$ **do**
3: $N \leftarrow BFS(G)$; ▷ Get next node in G
4: G_{cand} += $\{N\}$; G -= $\{N\}$; ▷ Update G_{cand}, G
5: **if** $t(G_{cand}) < t_{unit} - t_{delta}$ **then**
6: **continue**;
7: **else if** $t(G_{cand}) > t_{unit} + t_{delta}$ **or** $t(G_{cand}) + t(G) > t(G_{cand} + G) + t_{delta}$ **then**
8: Backtrack to another N during BFS, or accept the last N satisfying line 5 if none meets condition;
9: **else**
10: $\mathbb{G}.append(G_{cand})$; $G_{cand} \leftarrow \emptyset$; ▷ Get new piece

Jigsaw then carefully places BEV model components to overlap communication overhead. Precisely, FU should always be placed on the SoC executing the longest backbone and head components, which is illustrated in *step (2)* of Fig. 9. Additionally, several details are worth noticing when integrating with the sensing and planning stages. To avoid transferring large raw sensing data, the sensing stage of the distributed IB and PBs, including DSP pre-processing, should be separated into distinct SoCs. To facilitate the planning stage, it would be better to collect TH results onto a single SoC. Fortunately, the task results are more condensed compared to intermediate data. Their size is typically very small according to Sec. III-A.

C. Splitting Customization for PV Models

Performance interference happens when PV models run concurrently with the BEV model. To avoid interference, preemption at the end of each timeslot is necessary. However, current GPUs do not provide sound or user-controlled preemption due to problematicly large context switch overhead [29], [30]. To achieve controlled preemption, the *Splitting Customization* procedure splits PV models into fixed-size pieces as scheduling units before execution. The piece size is equal for the same timeslot and can be different for different timeslots. The piece size should be small to improve timeslot utilization, especially when there are multiple PV tasks with diverse execution time.

We make use of the graph structure of DNN models to find suitable splitting points. The structure of modern DNNs follows a layer chain with a handful of forked branches. Therefore, we can follow Algorithm 1 to split DNN models automatically into pieces of size t_{unit}. A splitting error up to $t_{delta} = 0.3$ ms is allowed in our implementation. Related implementation details are described in Section V-A.

D. Schedulability Analysis

Schedulability Analysis generates the feasible lower bound of all PV models' arrival periods. If PV tasks arrive too frequently, the timeslots would be unable to hold the PV models to satisfy their deadlines. The analysis of the stable and unstable timeslot is conducted independently. Note that the scheduler does not manage the BEV task. The BEV task's execution strategy is fixed in Sec. IV-B, and it would execute continuously without pause. No self-suspended task is involved during schedulability analysis.

Formally, we assume a given BEV model with BEV period T_{BEV}, stable timeslot size t_s and unstable timeslot size $\mathcal{T}_u$ (a random variable). There is a set of hard real-time PV tasks (go to stable timeslot) $\mathbb{T}_s = \{\tau_1^s, ..., \tau_m^s\}$ with execution time $\{C_1^s, ..., C_m^s\}$ and soft real-time tasks (go to unstable timeslot) $\mathbb{T}_u = \{\tau_1^u, ..., \tau_n^u\}$ with execution time $\{C_1^u, ..., C_n^u\}$. Schedulability analysis gives the feasible range of arrival periods $\{T_1^s, ..., T_m^s\}$ and $\{T_1^u, ..., T_n^u\}$ under selected piece size t_{unit}^s and t_{unit}^u. Tasks in $\mathbb{T}_u$ expect a deadline satisfaction rate above $p\%$. We follow the Earliest-Deadline-First (EDF) scheduling algorithm, which is optimal on preemptive uniprocessors [44].

Jigsaw requires an extra prerequisite on $T_1^s, ..., T_m^s$, $T_1^u, ..., T_n^u$: They must be integer multiples of T_{BEV}. This helps align the execution of PV and BEV tasks and facilitates schedulability analysis. This granularity limitation should be acceptable since $T_1^s, ..., T_m^s, T_1^u, ..., T_n^u$ are generally much larger than T_{BEV}. In addition, it assumes that the deadlines of PV tasks are at the boundary of BEV periods, after which the planning stage starts utilizing their results.

For $\mathbb{T}_s$, we extend existing research findings and derive a formula to check schedulability. It has been proven [44] that a necessary and sufficient schedulability condition under EDF for tasks in $\mathbb{T}$ when their deadlines are equal to periods is:

$$\sum_{\tau_i \in \mathbb{T}} \frac{C_i}{T_i} \leq 1 \tag{1}$$

where C_i/T_i is called the *utilization* of task τ_i^s. In our case, PV tasks in $\mathbb{T}_s$ can only execute in t_s one piece at a time. Therefore, we calculate it as the ratio of the following two terms: the number of pieces after splitting C_i^s, and the number of piece slots within all stable timeslots in T_i^s:

$$\sum_{\tau_i^s \in \mathbb{T}_s} \frac{\lceil C_i^s / t_{unit}^s \rceil}{T_i^s / T_{BEV} \times \lfloor t_s / t_{unit}^s \rfloor} \leq 1 \tag{2}$$

Since *Jigsaw* requires that PV tasks are aligned with BEV periods, they arrive only at BEV period boundaries, and the execution of their pieces strictly follows EDF without delay. This ensures the correctness of the formula above. The only tunable parameter in Equation (2) is t_{unit}^s. Generally, a very small t_{unit}^s exactly dividing t_s and every C_i would be optimal to avoid timeslot waste caused by rounding. However, splitting DNNs into small t_{unit}^s brings overhead in memory usage. Setting it to value exactly dividing t_s at around 3-5 ms would be good enough according to results in Sec. VI-C.

For $\mathbb{T}_u$, we replay a long enough PB latency trace to check schedulability. Since $\mathcal{T}_u$ follows a complex probability distribution depending on the input point cloud pattern that gradually shifts with time, it would be difficult and inaccurate

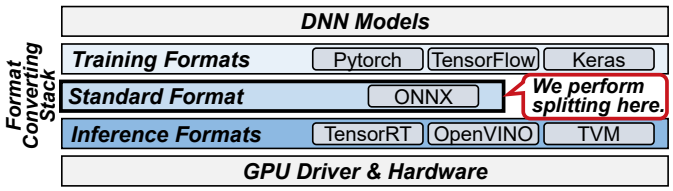

Fig. 10: Format converting stack before DNN deployment and our selected level to implement model splitting.

to model it mathematically. Therefore, we follow an easy-to-implement approach to check if the deadline satisfaction rate of $\mathbb{T}_u$ is above $p\%$ when scheduled on a long enough real trace of unstable timeslot size. This procedure can be accomplished by program simulation before actual deployment. Multiple $\mathbb{T}_u$ may exist when a BEV model contains multiple voxel-based PBs. In this case, the trace-replay approach still applies by treating the $\mathbb{T}_u$ on the same SoC uniformly. In our implementation, we abort an unfinished PV task at its deadline, giving way to the next iteration on fresh input data. According to our experiment, parameter p can be set at around 90, achieving a good schedulability boundary. The parameter t^u_{unit} should be set at a small value of around 2.5-5 ms to fully utilize the unstable timeslot (Section VI-C).

Schedulability analysis fails when a given set of PV tasks cannot be executed in timeslots under acceptable periods. In this case, we can make a compromise to enlarge the stable timeslot at the price of slowing down the BEV period by partial fallback to no parallelism. For the stable timeslot, we can disable the batch redistribution of a longer IB, place multiple backbones/heads on one SoC, or even completely run BEV and PV tasks on distinct SoCs. For the unstable timeslot, we can pad voxel-based PB longer than its tail latency.

Note that we assume all PV tasks are periodic with deadlines equal to periods. This covers the most common ADAS setup. For aperiodic tasks or tasks with deadlines smaller than periods, a stronger and more complex schedulability analysis is needed and one can resort to prior works [45], [46]. They require customization to be applicable for timeslot filling, which we leave for future work.

E. Stable and Unstable Timeslot Schedulers

Stable and Unstable Timeslot Schedulers are the sole modules integrated into the existing ADAS domain runtime. They are responsible for independently scheduling tasks in $\mathbb{T}_s$ and $\mathbb{T}_u$. Each of them only resides on the particular SoC with the corresponding timeslot. They receive requests periodically from PV tasks and schedule model pieces using EDF. Since schedulability analysis has passed, it can offer latency predictability for both BEV and PV tasks. Their detailed implementation is explained in Section V-B.

V. Implementation

A. Splitting DNN Models

To determine the proper level to implement model splitting, we need to examine the whole software framework stack

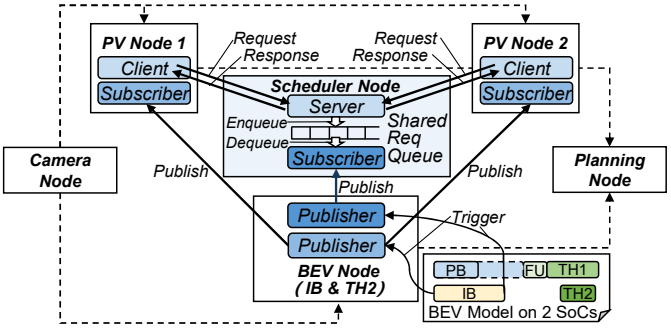

Fig. 11: Implementation of stable timeslot scheduler on SoC 2 based on ROS 2. Blue modules show scheduler building blocks.

when a DNN model gets optimized, which is demonstrated in Fig. 10. Models written and trained with PyTorch [47], Tensorflow [48], or other user-friendly *training frameworks* are first exported to a *standard representation* and then optimized by specialized *inference frameworks* for up to several times speedup. The optimization result is a series of efficient GPU kernels that can be submitted to the GPU driver by inference framework runtime for final execution.

Implementing splitting at the training format level does not produce a generic solution since each popular framework would require a specialized implementation. A seemingly attractive choice is to conduct splitting at the inference framework level by dividing their generated kernels into groups. However, this also requires specialized and complicated hack-style modifications in their runtime. Furthermore, some vendor-specific inference frameworks are not open-sourced and cannot be modified, including Intel OpenVINO [49], NVIDIA TensorRT [50], etc. Therefore, we choose to split PV models at the standard representation level, which has only one popular ONNX format [51] and is regarded as the "narrow waist" in the whole stack [52]. This format describes a model as a graph, which is suitable for Alg. 1.

B. Integrating Schedulers into ADAS Domain

To correctly integrate the scheduler modules into the ADAS domain, it is necessary to understand its current organization. In practice, ADAS domain workload is organized in a data-driven manner by *communication middleware* (e.g., ROS [53], ROS 2 [54], CyberRT [55]) on top of the operating system. It breaks down the whole workload into many nodes and defines dataflow by specifying publish-subscribe or request-response relationships among nodes. The computation of each node is triggered independently on the arrival of a subscribed message or a service request. The publish-subscribe relationship is used primarily. For instance, perception node A for BEV task and another perception node B for traffic light detection can subscribe to the camera sensing node, while the planning node subscribes to both node A and B. Multiple nodes can execute simultaneously, thus forming a pipeline. A node can ignore

TABLE I: Details of four selected BEV models.

Model Name	Description
BEV Model 1 (pillar-based, small)	Resnet50 [56] 256x704 + PointPillars [40] 0.25 + BEVFusion [15] Fuser 100x100 + GenRes Seg Head [14] + SSD Det Head [57]
BEV Model 2 (pillar-based, medium)	Resnet101 [56] 256x704 + PointPillars [40] 0.2 + BEVFusion [15] Fuser 128x128 + GenRes Seg Head [14] + SSD Det Head [57]
BEV Model 3 (voxel-based, medium)	Resnet50 [56] 256x704 + VoxelNet [25] 0.1 + BEVFusion [15] Fuser 128x128 + SECOND Seg Head [26] + Transfusion Det Head [58]
BEV Model 4 (voxel-based, large)	SwinT [59] 256x704 + VoxelNet [25] 0.075 + BEVFusion [15] Fuser 180x180 + SECOND Seg Head [26] + Transfusion Det Head [58]

some upstream data if it comes too frequently. The actual carrier of a node can be either a process or a thread.

We implement the two schedulers as two specialized nodes. The detail of the stable timeslot scheduler on SoC 2 is demonstrated in Fig. 11. Instead of executing DNN models by themselves, PV nodes periodically send execution requests to the scheduler node and wait for its response. The scheduler enqueues received requests and waits for the next stable timeslot. This is achieved by subscribing to a specialized publisher in the BEV node, which is triggered whenever IB finishes. On receiving the subscribed message, the scheduler dispatches a fixed number of model pieces for pending PV tasks, prioritizing the one with the earliest deadline. A pending task gets dequeued when its last model piece finishes execution, and the scheduler then sends a response back to the corresponding PV node. The BEV node also acts as a timer by publishing a ticking message at every BEV period boundary. PV nodes subscribe to it and send execution requests every several BEV periods. In our implementation, all nodes are bound to isolated CPU cores. The unstable timeslot scheduler is identical to Fig. 11, which dispatches a varying number of PV model pieces after voxel-based PBs.

In summary, our schedulers integrate into the ADAS domain using the interface provided by communication middleware with minimal modification on PV and BEV nodes.

VI. Evaluation

A. Evaluation Setup

Workloads. We conduct all experiments using the four BEV model variants listed in Tab. I. Each of them has a LiDAR PB, a camera IB, an object detection task head (DH), and a scene segmentation task head (SH). We follow the two most popular BEV model papers [14], [15] as well as their industrial and academic implementation [60]–[62] to build four model variants with distinct tradeoff in latency and model complexity (accuracy). The variants differ in component choices, input image resolution, input point cloud resolution (voxel size), and BEV space resolution. Only BEV models 3 and 4 have unstable timeslot since they use voxel-based PB. We use the Nuscenes dataset [37], and its total camera batch size is 6. For PV models, we choose YOLOv4x [63] and Segformer [64] for illustration. They are widely used for object detection and scene segmentation respectively. Their input image resolutions are set at 512x1408, and their execution time is around 24 and 45 ms on our testbed.

In our evaluation, all DNN models are written in PyTorch 1.12.1, converted to ONNX 1.15.0, and optimized by TensorRT 8.5.2 in FP16. One exception is voxel-based PB, whose sparse operators are not supported by TensorRT. We optimize it by NVIDIA libspconv [65]. We use ROS 2 version Foxy to organize the perception workload.

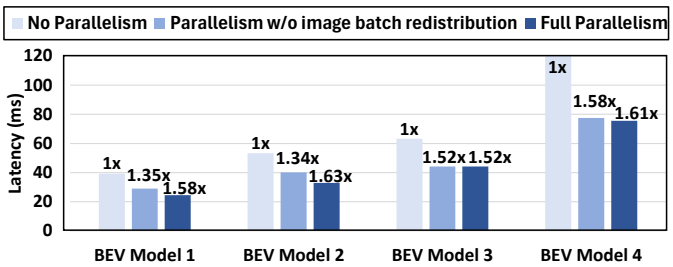

Fig. 12: Parallel speedup of BEV models in different setups.

Testbed. We build a connected dual-SoC platform using two NVIDIA Jetson AGX Orins, each equipped with a 12-core ARM CPU, an NVIDIA Ampere GPU, and 32 GB LPDDR5 shared memory. We attach NVIDIA Mellanox ConnectX-5 network adapter cards with RDMA support on their PCIe slots, providing up to 50 Gb/s bandwidth under PCIe 3.0×8. The Orins are both running Jetpack SDK 5.1.1 containing Linux kernel 5.10, Ubuntu 20.04, and CUDA 11.4. The Orins are set at maximum power mode (*MAXN*), and their GPU frequencies are fixed at the highest.

B. Mechanism Effectiveness

In this section, we check the effectiveness of the two core mechanisms in *Jigsaw* framework: *accelerating BEV model by exploiting component parallelism*, and *guaranteeing latency predictability by filling PV models into timeslots*.

Parallel speedup. We measure the latency when executing BEV model components under three setups: sequentially on a single machine (*No Parallelism*), in parallel following Section IV-B but without image batch redistribution (*Parallelism w/o image batch redistribution*) and in parallel completely following Section IV-B (*Full Parallelism*). The BEV model latency and the corresponding speedup compared to *No Parallelism* are shown in Fig. 12. Although the ideal yet unreachable speedup would be 2×, *Full Parallelism* still achieves a considerable speedup of 1.52-1.63×. *Parallelism w/o image batch redistribution* can also achieve 1.34-1.58× speedup. Since PB is longer than IB for BEV model 3, the *Parallelism w/o image batch redistribution* and *Full Parallelism* bars are the same. By comparing the results with *Full Parallelism*, we can see the effectiveness of image batch redistribution, especially for BEV models 1 and 2 with pillar-based PB.

No interference. *Jigsaw* guarantees no performance interference among the BEV and PV tasks. Fig. 13 shows the cumulative distribution function (CDF) curve of the parallelized BEV task's latency when executing with PV model YOLOv4x or Segformer. We contrast our proposed scheduling mechanism with the native scheduling mechanisms offered by NVIDIA GPU, including MultiProcess and MultiStream introduced in Section III-C. In MultiStream case, we set the BEV task at a higher priority than the PV task. The arrival

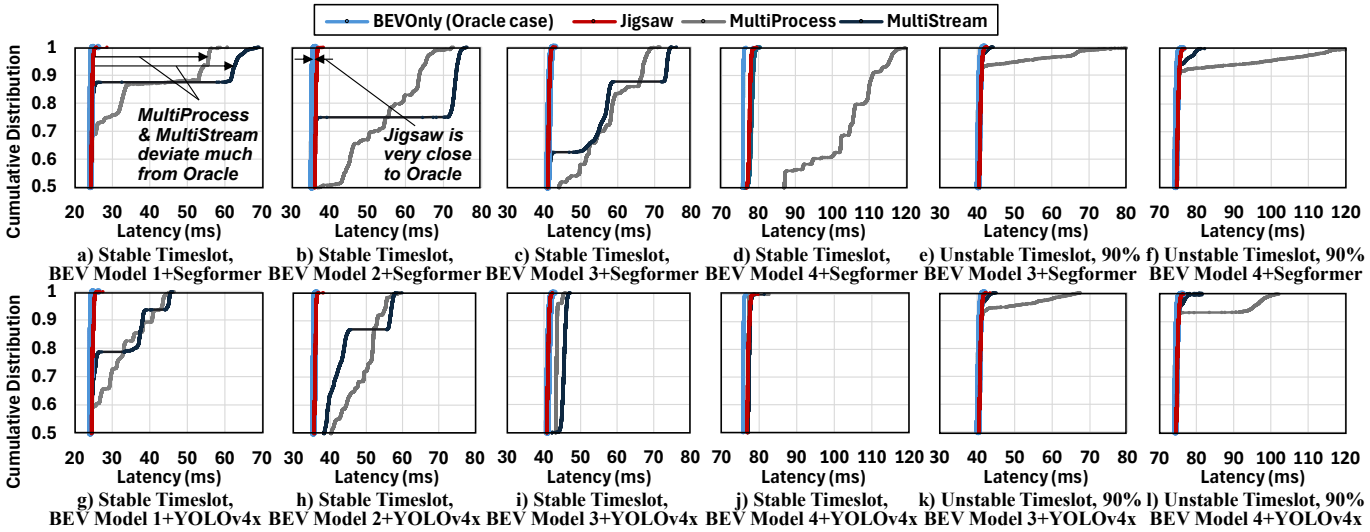

Fig. 13: Cumulative distribution curve of BEV task latency when co-running with PV tasks.

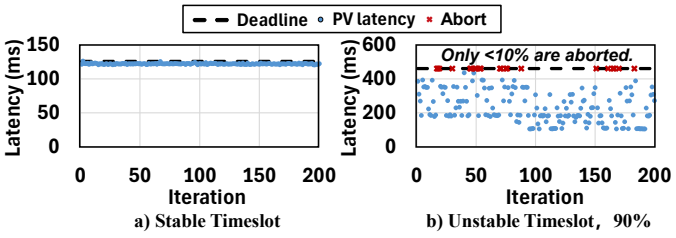

Fig. 14: Latency timeline of PV task using different timeslots.

period of the PV task is set at a proper value such that the total available timeslot length within the period is just enough to accommodate all split PV model pieces when using our mechanism. The size of PV model pieces or t_{unit} is set at 5 ms. The latency is measured on the Nuscenes dataset [37] with 6019 samples. As for comparison, we also show the CDF curve when the BEV task is executing alone (BEVOnly).

In Fig. 13(a)-(d), (g)-(j), the chosen PV task arrives periodically on SoC 2 having the stable timeslot. Under MultiProcess and MultiStream, the tail latency of the BEV task can increase to a large extent, especially for BEV models 1 and 2 whose stable timeslots are small. Although MultiStream could postpone the deviation point of the curve from BEVOnly compared to MultiProcess, it does not help with tail latency. One may notice that in Fig. 13(i)(j), the tail latency under native scheduling mechanisms does not deviate much from BEVOnly. This is because the stable timeslot of BEV models 3 and 4 is close to or even larger than the execution time of YOLOv4x. Only in this case could the native scheduler function well since PV models are essentially filling the stable timeslot as our scheduling mechanism does.

In Fig. 13(e)(f)(k)(l), the chosen PV task arrives periodically on SoC 1 having the unstable timeslot with expected deadline satisfaction rate $p\% = 90\%$ in our mechanism. MultiProcess still causes a significant increase in BEV task tail latency while MultiStream only slightly increases it. We believe the reason lies in the special sparse operators used by PB. Sparse operators cannot fully utilize the GPU hardware resource since they involve complicated control logic [66], [67]. Therefore, MultiStream mechanism could enable simultaneous kernel execution to accommodate PV tasks. We admit the limitation of our mechanism in the incapability to utilize this phenomenon. However, our mechanism provides a unified solution for both types of timeslots and presents a stable BEV task latency CDF curve very close to BEVOnly under all cases.

Deadline Satisfaction. Fig. 14 showcases the execution timeline of PV task Segformer running with BEV model 3 utilizing stable and unstable timeslot, respectively (same setup as Fig. 13(c)(e)). We show 200 iterations due to space limitations. In Fig. 14(a), it is clear that our scheduling mechanism ensures stable PV task latency within its deadline using the stable timeslot. In Fig. 14(b), our mechanism achieves about 92% deadline satisfaction rate using the unstable timeslot for the soft-real time PV task expecting 90% deadline satisfaction. It is worth mentioning that the PV task's deadlines are set at the minimum available values following the schedulability analysis procedure introduced in Section IV-D.

C. Detailed Analysis

In this section, we dive into the details of *Jigsaw* to answer two important questions: *Under what scope do the proposed mechanisms apply? How much overhead do the proposed mechanisms introduce?*

Schedulability boundary. The prerequisite for latency predictability is passing the schedulability check. In this part, we visually demonstrate the lower bound of the PV tasks' periods when scheduled with the BEV models in Tab. I. Parameter setting is also discussed.

Special case. We first study a special yet clear and instructive case when the period of all hard/soft real-time PV tasks are respectively equal, or $T_i^s|_{\tau_i^s \in \mathbb{T}_s} = T_s$ and $T_i^u|_{\tau_i^u \in \mathbb{T}_u} = T_u$.

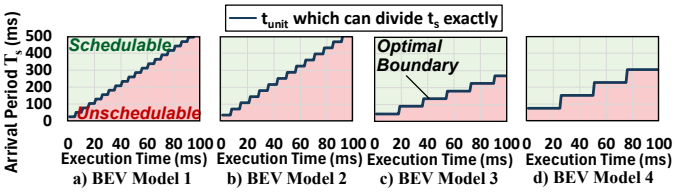

Fig. 15: Schedulability boundary using stable timeslot when multiple PV tasks share one same period.

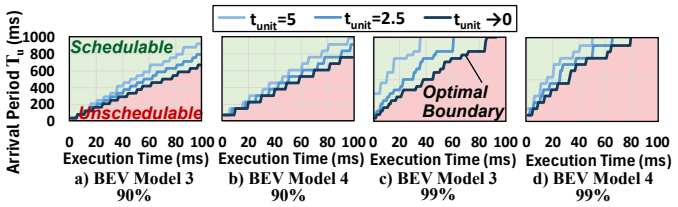

Fig. 16: Schedulability boundary using unstable timeslot when multiple PV tasks share one same period.

For the stable timeslot, the schedulability condition becomes

$$\frac{\left\lceil \sum_{\tau_i^s \in \mathbb{T}_s} C_i^s / t_{unit}^s \right\rceil}{T_s / T_{BEV} \times \lfloor t_s / t_{unit}^s \rfloor} \leq 1 \tag{3}$$

Fig. 15 shows the schedulability boundaries of T_s depending on $\sum_{\tau_i^s \in \mathbb{T}_s} C_i^s$ following Equation (3). In general, the schedulable region is large enough to hold PV tasks with less than 100 ms total execution time at more than 2-3 FPS. The boundaries are in step-like shape since we require the PV models' period to be integer multiples of T_{BEV} (Sec. IV-D). The slope of the boundaries is determined by the relative size of stable timeslot t_s/T_{BEV}. The step granularity depends on the length of BEV period T_{BEV}. In this case, setting the parameter t_{unit}^s to any value exactly dividing t_s can reach the optimal boundary where the timeslot is fully utilized.

For unstable timeslot, Fig. 16 shows the schedulability boundaries of T_u depending on $\sum_{\tau_i^u \in \mathbb{T}_u} C_i^u$ obtained by trace replaying at $p = 90$ and 99. In general, the schedulable region can hold PV tasks with less than 100 ms total execution time at more than ∼1 FPS. This is less spacious compared to the stable timeslot. For unstable timeslot, the parameter t_{unit}^u would be set as small as possible for full utilization (optimal). However, DNN models could not be split infinitely. Therefore, we show the boundaries when t_{unit}^u is set at reasonably small values, including 5 ms and 2.5 ms. They are both close to optimal when $p = 90$ but suffer from large deviation when $p = 99$. This indicates that p should not be set too high.

General case. We then consider the general case when the period of PV tasks can take diverse values. In this case, the schedulability boundary for any given $\mathbb{T}_s$ and $\mathbb{T}_u$ spans into high-dimensional space whose dimension is equal to the cardinality or size of $\mathbb{T}_s$ and $\mathbb{T}_u$ respectively. For visualization, we plot the two-dimensional boundaries using Segformer and YOLOv4x as PV tasks for visualization. When the number of PV tasks gets larger, *Jigsaw* users could still plot 2D

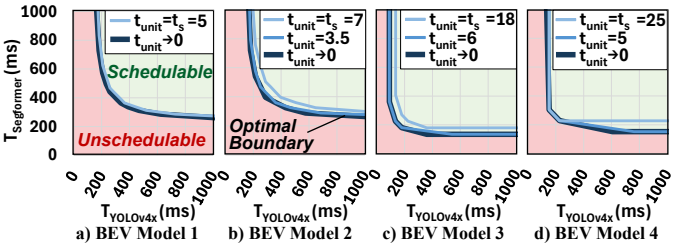

Fig. 17: Schedulability boundary using stable timeslot when two PV tasks have distinct periods.

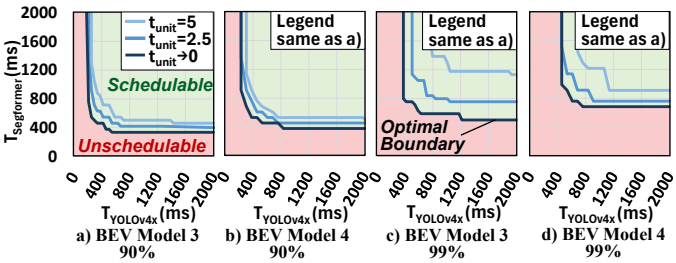

Fig. 18: Schedulability boundary using unstable timeslot when two PV tasks have distinct periods.

projections of the boundary to determine period settings. In practice, there would not be many PV tasks, and multiple PV tasks can be classified into several period levels.

Fig. 17 plots the boundaries using stable timeslot. They form polylines near the hyperbola specified by Equation (2) since we require $\{T_1, ..., T_n\}$ to be integer multiples of T_{BEV}. Ideally, t_{unit}^s should be set to a very small value which could both exactly divide t_s and all $C_i^s|_{\tau_i^s \in \mathbb{T}_s}$. This forms the optimal boundary fully utilizing the stable timeslot. However, a very small t_{unit}^s might not be possible due to splitting overhead. Therefore, we investigate several small enough t_{unit}^s around 3-5 ms exactly dividing t_s. The resulting boundaries are already very close to the optimal one.

Fig. 18 plots the boundaries using unstable timeslot. The boundaries are obtained by trace replaying at $p = 90$ and 99. The schedulable region is smaller than using the stable timeslot. Setting t_{unit}^u at around 2.5-5 ms when $p = 90$ yields good boundaries close to the optimal one.

Mechanism overhead. Executing a PV model in pieces could bring extra overhead compared to executing it as a whole. We investigate the overhead in terms of total execution time and total memory usage when $t_{unit} = 10$, 5, and 2.5 ms. Results are shown in Fig. 19. In general, the total execution time increases slightly in Fig. 19(a) since the splitting might prevent some layer-fusion optimization chances when using inference frameworks. However, sometimes, the splitting helps reduce total execution time ($t_{unit} = 5$, Segformer). This is because inference frameworks work by searching with heuristics, and the split may affect its behavior to generate slightly better search results.

Splitting increases the total memory usage of a PV model by 0.31-1.08 GB when $t_{unit} = 2.5$ ms as shown in Fig. 19(b).

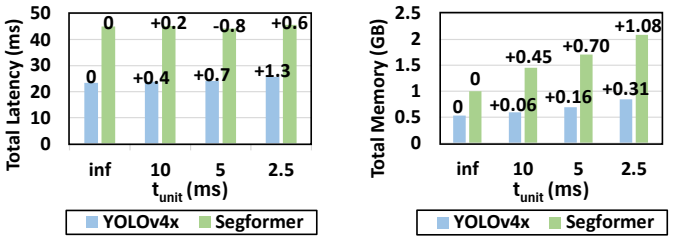

Fig. 19: DNN splitting overhead in latency and memory usage.

However, this overhead can be eliminated with minor modifications to inference framework APIs. After splitting, the underlying inference framework allocates separate memory space for each model piece. The memory can essentially be reused since the pieces are executed one by one. For closed-sourced frameworks as we use, the main memory size of widely used SoC can be 32-64 GB, which could bear the overhead for a handful of PV tasks in practice.

As for the schedulers, each of them occupies one CPU core to ensure responsiveness in our implementation for simplicity. In practice, granting the highest operating system priority to the process carrying the scheduler node should be fine.

VII. Discussion

Other factors affecting predictability. Shared-memory SoCs are well-known to have memory bandwidth contention among execution units [10], [68]. In theory, DSP and CPU workloads in the sensing and planning stage may affect GPU performance. However, their bandwidth appears small compared to DNN inference tasks and causes minimal contention. A NVIDIA Orin can offer 204 GB/s memory bandwidth. The BEV DNN inference tasks on GPU can cause $10-15\%$ total available bandwidth. Ten cameras working at 1920×1080 30 FPS causes $< 1\%$ total available bandwidth. Ten LiDARs used by Nuscenes working at 30 Hz cause $< 0.2\%$ total available bandwidth. The CPU planning tasks are not memory intensive and use very small bandwidth.

Applicability to other platform architectures. *Jigsaw* mainly targets the most popular dual-SoC platform, each equipped with a GPU. If necessary, *Jigsaw* can be extended to more than two SoCs. Component parallelism still applies, while the number of stable/unstable timeslots would increase. In this case, *Jigsaw* could build a PV task scheduler for each type of timeslot on each SoC. It has to conduct schedulability analysis under every possible task-timeslot mapping to find a satisfiable $T_1^s, ..., T_m^s$, $T_1^u, ..., T_n^u$. In practice, an ADAS domain controller with many SoCs is less seen due to more complex interconnection topology and higher cost. The parallelism degree of BEV-centric perception is also limited, which makes using too many SoCs unnecessary.

Jigsaw is also suitable for SoCs equipped with ASIC AI accelerators instead of GPUs, which is seen in non-NVIDIA platforms [69]–[71]. AI accelerators generally only allow sequential task execution without preemption support. This is compatible with *Jigsaw* since it bypasses concurrent GPU multi-tasking and performs preemption manually. Some platforms have multiple AI accelerators in one SoC [7]. *Jigsaw* is applicable to them as a multi-SoC with super-fast interconnect.

Some SoCs are equipped with both GPU and AI accelerators [7], [72]. However, only one of them is the major execution unit. For example, the NNA accelerator is the major unit of Tesla FSD 3, whose GPU is much weaker and is only responsible for a few NNA-non-supported operations [7]. The GPU is the major unit of NVIDIA Orin, whose NVDLA accelerator can only execute specific convolutional IB models. It is also much slower than GPU (3-4× according to our measurement) and is recommended for power-saving use cases [72]. Considering this and avoiding being overly platform-specific, our description of *Jigsaw* does not use the NVDLA. Partially offloading IB to the NVDLA for further latency reduction is applicable and is orthogonal to *Jigsaw*'s management of the major DNN execution unit, GPU.

VIII. Related Work

ADAS system analysis. In the past few years, people have brought comprehensive review [1], [73], [74] and introductory papers [3], [5], [75] on the computing architecture of ADAS domain of the day. However, only a few have witnessed the very recent BEV-centric perception paradigm [13] and the prevalence of dual-SoC platforms [73], [76].

Notably, some prior works focus on the timing analysis of ADAS domain. Miguel et al. model the uncertain latency mathematically with random variables [77]. Voronov et al. extend traditional graph response-time analysis to address hardware heterogeneity [78]. These works are valuable for the whole ADAS workload pipeline involving various tasks, while *Jigsaw* focuses on organizing the most compute-intensive GPU perception tasks.

DNN parallel speedup. Existing DNN parallel speedup mechanisms can be divided into three categories. *Jigsaw*'s parallelism approach lies between data parallelism and inter-layer parallelism. It exploits the BEV model's structure, distributing independent components and their input data to reduce latency. The details of the three mechanisms are described below.

Data Parallelism suggests distributing input data batches to multiple devices executing the same DNN model. It is mostly used during model training where a large number of input data is involved [79]. In real-time cases with streaming data, conventional data parallelism is not very suitable.

Inter-layer Parallelism suggests vertically slicing a model at layer boundary and distributing slices, forming a pipeline to improve throughput. It is widely used to speedup Large Language Model (LLM) training in datacenter [80] or to utilize heterogeneous hardwares on a single device [81], [82]. However, conventional inter-layer parallelism would incur data transfer overhead in the middle and increase total latency.

Intra-layer Parallelism suggests horizontally slicing weight and input tensors of available model layers and distributing slices to reduce latency. It is applied to speedup LLM training/inference in datacenter [83], [84] or to enable collaborative edge CNN inference on embedded single-board computers

(SBC) [85]–[87]. However, it requires very frequent data synchronization after each sliced layer. This would bring huge overhead for the dual-SoC platform. Furthermore, sparse operators used in PB are not supported.

GPU scheduling. The native GPU multi-tasking mechanism is shown to be deficient in latency-critical cases [27], [28]. Researchers have proposed two major improvement approaches.

One approach aims at enabling fast GPU preemptive execution. They either extend the hardware [88], [89], enhance the driver [90], [91] or customize the inference framework [92], [93]. In ADAS setup, REEF performs cross-layer software optimization for micro-second scale preemption [94]. However, they all require costly or complex modifications. Consequently, no commodity GPU offers sound preemption support yet.

The other approach builds customized workload management mechanisms. *Jigsaw* lies in this category. In ADAS setup, many works assume simultaneous multi-DNN inference on one large server-class GPU and propose fine-grained stream synchronization approaches [20]–[22]. However, the SoC-integrated GPU is less capacious, and fine-grained stream management is generally not possible when using inference frameworks. Some works optimize multiple convolutional image DNNs on a heterogeneous platform with both GPU and AI accelerators [95]–[98]. D3 [4] questions the traditional data-driven organization (e.g., ROS 2) and proposes a deadline-driven one for improved deadline satisfaction. Contrarily, our scheduler follows the ROS 2. Prophet [12] predicts latency variation of some ADAS perception DNNs and coordinates them for predictability by early-exit. However, it is designed for neither the modern BEV-centric perception nor the dual-SoC hardware setup. Clockwork [99] and Paella [100] build schedulers for cloud datacenters providing QoS guarantee. *Jigsaw* adopts their sequential execution concept in ADAS setup and further splits large DNN models for controlled preemption.

IX. Conclusion

This paper proposes *Jigsaw*, a task management framework deploying BEV-centric perception workload on a dual-SoC platform under real-time requirements. It integrates two core mechanisms to first exploit *component parallelism* to reduce the BEV model latency and then schedule PV models by *time-slot filling* to ensure latency predictability. Evaluation results show its effectiveness under a practical dual-SoC platform equipped with NVIDIA GPUs and connected via a PCIe bus.

X. Acknowledgement

We sincerely thank all the anonymous reviewers for their valuable feedback. This work is supported by the National Natural Science Foundation of China (No. U23A6007) and the STCSM International S&T Cooperation Project (23510713200). The corresponding author is Chao Li.

References

[1] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.

[2] F. Jiménez, J. E. Naranjo, J. J. Anaya, F. García, A. Ponz, and J. M. Armingol, "Advanced driver assistance system for road environments to improve safety and efficiency," *Transportation research procedia*, vol. 14, pp. 2245–2254, 2016.

[3] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 751–766.

[4] I. Gog, S. Kalra, P. Schafhalter, J. E. Gonzalez, and I. Stoica, "D3: a dynamic deadline-driven approach for building autonomous vehicles," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 453–471.

[5] B. Yu, W. Hu, L. Xu, J. Tang, S. Liu, and Y. Zhu, "Building the computing system for autonomous micromobility vehicles: Design constraints and architectural optimizations," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 1067–1081.

[6] NVIDIA, "Nvidia drive px 2 archive," https://developer.nvidia.com/drive/px2.

[7] E. Talpes, D. D. Sarma, G. Venkataramanan, P. Bannon, B. McGee, B. Floering, A. Jalote, C. Hsiong, S. Arora, A. Gorti *et al.*, "Compute solution for tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, 2020.

[8] NVIDIA, "Non-transparent bridging and pcie interface communication," https://docs.nvidia.com/drive/drive-os-5.2.0.0L/drive-os/index.html#page/DRIVE_OS_Linux_SDK_Development_Guide/Interfaces/sys_components_non_transparent_bridging.html.

[9] Microchip, "Switchtec pcie switches," https://www.microchip.com/en-us/products/interface-and-connectivity/pcie-switches.

[10] Y. Xu, M. E. Belviranli, X. Shen, and J. Vetter, "Pccs: Processor-centric contention-aware slowdown model for heterogeneous system-on-chips," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1282–1295.

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

[12] 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)*. IEEE, 2022, pp. 305–317.

[13] H. Li, C. Sima, J. Dai, W. Wang, L. Lu, H. Wang, J. Zeng, Z. Li, J. Yang, H. Deng *et al.*, "Delving into the devils of bird's-eye-view perception: A review, evaluation and recipe," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[14] Z. Liu, H. Tang, A. Amini, X. Yang, H. Mao, D. L. Rus, and S. Han, "Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 2774–2781.

[15] T. Liang, H. Xie, K. Yu, Z. Xia, Z. Lin, Y. Wang, T. Tang, B. Wang, and Z. Tang, "Bevfusion: A simple and robust lidar-camera fusion framework," *Advances in Neural Information Processing Systems*, vol. 35, pp. 10 421–10 434, 2022.

[16] J. H. Yoo, Y. Kim, J. Kim, and J. W. Choi, "3d-cvf: Generating joint camera and lidar features using cross-view spatial feature fusion for 3d object detection," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII 16*. Springer, 2020, pp. 720–736.

[17] 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.

[18] S. Borse, M. Klingner, V. R. Kumar, H. Cai, A. Almuzairee, S. Yogamani, and F. Porikli, "X-align: Cross-modal cross-view alignment for bird's-eye-view segmentation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 3287–3297.

[19] Y. Li, A. W. Yu, T. Meng, B. Caine, J. Ngiam, D. Peng, J. Shen, Y. Lu, D. Zhou, Q. V. Le *et al.*, "Deepfusion: Lidar-camera deep fusion for multi-modal 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 17 182–17 191.

[20] 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). IEEE, 2018, pp. 190–201.

[21] F. Yu, S. Bray, D. Wang, L. Shangguan, X. Tang, C. Liu, and X. Chen, "Automated runtime-aware scheduling for multi-tenant dnn inference on gpu," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[22] M. Yang, S. Wang, J. Bakita, T. Vu, F. D. Smith, J. H. Anderson, and J.-M. Frahm, "Re-thinking cnn frameworks for time-sensitive autonomous-driving applications: Addressing an industrial challenge," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 305–317.

[23] S. Bateni and C. Liu, "Neuos: A latency-predictable multi-dimensional optimization framework for dnn-driven autonomous systems," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 371–385.

[24] C. Wan, M. Santriaji, E. Rogers, H. Hoffmann, M. Maire, and S. Lu, "Alert: Accurate learning for energy and timeliness," in *2020 USENIX annual technical conference (USENIX ATC 20)*, 2020, pp. 353–369.

[25] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4490–4499.

[26] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[27] M. Yang, "Avoiding pitfalls when using nvidia gpus for real-time tasks in autonomous systems," in *Proceedings of the 30th Euromicro Conference on Real-Time Systems*, 2018.

[28] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, "Gpu scheduling on the nvidia tx2: Hidden details revealed," in *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2017, pp. 104–115.

[29] 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.

[30] 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.

[31] K. Gadzicki, R. Khamsehashari, and C. Zetzsche, "Early vs late fusion in multimodal convolutional neural networks," in *2020 IEEE 23rd international conference on information fusion (FUSION)*. IEEE, 2020, pp. 1–6.

[32] Z. Zhang, M. Xu, W. Zhou, T. Peng, L. Li, and S. Poslad, "Bev-locator: An end-to-end visual semantic localization network using multi-view images," *arXiv preprint arXiv:2211.14927*, 2022.

[33] M. Tech, "Miivii apex dual orin - miivii technology," https://en.miivii.com/index.php?s=index/category/index&id=144.

[34] ResearchInChina, "Autonomous driving soc research report, 2023," http://www.researchinchina.com/Htmls/Report/2023/72878.html.

[35] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[36] Baidu, "Apollo home page," https://www.apollo.auto/.

[37] 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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 621–11 631.

[38] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[39] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-voxel cnn for efficient 3d deep learning," *Advances in neural information processing systems*, vol. 32, 2019.

[40] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 12 697–12 705.

[41] NVIDIA, "Cuda toolkit documentation," https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__STREAM.html#group__CUDART__STREAM_1ge2be9e9858849bf62ba4a8b66d1c3540.

[42] ——, "Nvidia multi process service," https://docs.nvidia.com/deploy/mps/.

[43] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A review of yolo algorithm developments," *Procedia computer science*, vol. 199, pp. 1066–1073, 2022.

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

[45] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with edf scheduling," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1250–1258, 2009.

[46] M. Short, "Improved schedulability analysis of implicit deadline tasks under limited preemption edf scheduling," in *ETFA2011*. IEEE, 2011, pp. 1–8.

[47] 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.

[48] 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.

[49] Intel, "Openvino toolkit," https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html.

[50] NVIDIA, "Tensorrt sdk," https://developer.nvidia.com/tensorrt.

[51] J. Bai, F. Lu, K. Zhang *et al.*, "Onnx: Open neural network exchange," https://github.com/onnx/onnx, 2019.

[52] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang *et al.*, "Mlperf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, 2020.

[53] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[54] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[55] Baidu, "Apollo cyberrt framework for autonomous driving," https://github.com/storypku/CyberRT.

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[57] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[58] X. Bai, Z. Hu, X. Zhu, Q. Huang, Y. Chen, H. Fu, and C.-L. Tai, "Transfusion: Robust lidar-camera fusion for 3d object detection with transformers," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 1090–1099.

[59] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[60] NVIDIA, "Cuda-bevfusion," https://github.com/NVIDIA-AI-IOT/Lidar_AI_Solution/tree/master/CUDA-BEVFusion.

[61] O. D. Team, "Openpcdet: An open-source toolbox for 3d object detection from point clouds," https://github.com/open-mmlab/OpenPCDet, 2020.

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

[63] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[64] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers," *Advances in neural information processing systems*, vol. 34, pp. 12 077–12 090, 2021.

[65] NVIDIA, "3dsparseconvolution," https://github.com/NVIDIA-AI-IOT/Lidar_AI_Solution/tree/master/libraries/3DSparseConvolution.

[66] B. Graham, M. Engelcke, and L. Van Der Maaten, "3d semantic segmentation with submanifold sparse convolutional networks," in

Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 9224–9232.

[67] Y. Lin, Z. Zhang, H. Tang, H. Wang, and S. Han, "Pointacc: Efficient point cloud accelerator," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 449–461.

[68] Q. Zhu, B. Wu, X. Shen, L. Shen, and Z. Wang, "Co-run scheduling with power cap on integrated cpu-gpu systems," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 967–977.

[69] Horizon, "Horizon journey series," https://en.horizon.cc/horizon-journey-series/.

[70] Mobileye, "Eyeq kit," https://www.mobileye.com/solutions/eyeq-kit/.

[71] Qualcomm, "Snapdragon ride," https://www.qualcomm.com/products/automotive/automated-driving.

[72] NVIDIA, "Nvidia Drive AGX Orin," https://www.nvidia.cn/self-driving-cars/drive-platform/hardware/.

[73] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6469–6486, 2020.

[74] D. Bastos, P. P. Monteiro, A. S. Oliveira, and M. V. Drummond, "An overview of lidar requirements and techniques for autonomous driving," in *2021 Telecoms Conference (ConfTELE)*. IEEE, 2021, pp. 1–6.

[75] H.-H. Sung, Y. Xu, J. Guan, W. Niu, B. Ren, Y. Wang, S. Liu, and X. Shen, "Brief industry paper: Enabling level-4 autonomous driving on a single $1k off-the-shelf card," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2022, pp. 297–300.

[76] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.

[77] M. Alcon, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. J. Cazorla, "Timing of autonomous driving software: Problem analysis and prospects for future solutions," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 267–280.

[78] S. Voronov, S. Tang, T. Amert, and J. H. Anderson, "Ai meets real-time: Addressing real-world complexities in graph response-time analysis," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 82–96.

[79] W. Xu, Y. Zhang, and X. Tang, "Parallelizing dnn training on gpus: Challenges and opportunities," in *Companion Proceedings of the Web Conference 2021*, 2021, pp. 174–178.

[80] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.

[81] Y. Hu, C. Imes, X. Zhao, S. Kundu, P. A. Beerel, S. P. Crago, and J. P. N. Walters, "Pipeline parallelism for inference on heterogeneous edge computing," *arXiv preprint arXiv:2110.14895*, 2021.

[82] 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)*. IEEE, 2019, pp. 392–405.

[83] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[84] S. Wang, J. Wei, A. Sabne, A. Davis, B. Ilbeyi, B. Hechtman, D. Chen, K. S. Murthy, M. Maggioni, Q. Zhang *et al.*, "Overlap communication with dependent computation via decomposition in large deep learning models," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2022, pp. 93–106.

[85] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1396–1401.

[86] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[87] S. Zhang, S. Zhang, Z. Qian, J. Wu, Y. Jin, and S. Lu, "Deepslicing: collaborative and adaptive cnn inference with low latency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2175–2187, 2021.

[88] N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru, "Deadline-based scheduling for gpu with preemption support," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 119–130.

[89] Z. Lin, L. Nyland, and H. Zhou, "Enabling efficient preemption for simt architectures with lightweight context switching," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 898–908.

[90] C. Basaran and K.-D. Kang, "Supporting preemptive task executions and memory copies in gpgpus," in *2012 24th Euromicro Conference on Real-Time Systems*. IEEE, 2012, pp. 287–296.

[91] H. Zhou, G. Tong, and C. Liu, "Gpes: A preemptive execution system for gpgpu computing," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2015, pp. 87–97.

[92] G. Chen, Y. Zhao, X. Shen, and H. Zhou, "Effisha: A software framework for enabling effficient preemptive scheduling of gpu," in *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2017, pp. 3–16.

[93] B. Wu, X. Liu, X. Zhou, and C. Jiang, "Flep: Enabling flexible and efficient preemption on gpus," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 483–496, 2017.

[94] 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)*, 2022, pp. 539–558.

[95] L. Sun, X. Hou, C. Li, J. Liu, X. Wang, Q. Chen, and M. Guo, "A2: Towards accelerator level parallelism for autonomous micromobility systems," *ACM Transactions on Architecture and Code Optimization*, 2024.

[96] X. Wang, C. Li, L. Sun, Q. Lv, X. Hou, J. Leng, and M. Guo, "Psheeo: Continuous energy efficiency optimization in autonomous embedded systems," in *2024 IEEE 42st International Conference on Computer Design (ICCD)*, 2024.

[97] D. Chun, J. Choi, H.-J. Lee, and H. Kim, "Cp-cnn: computational parallelization of cnn-based object detectors in heterogeneous embedded systems for autonomous driving," *IEEE Access*, vol. 11, pp. 52 812–52 823, 2023.

[98] I. Dagli, A. Cieslewicz, J. McClurg, and M. E. Belviranli, "Axonn: Energy-aware execution of neural network inference on multi-accelerator heterogeneous socs," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1069–1074.

[99] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving dnns like clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 443–462.

[100] K. K. Ng, H. M. Demoulin, and V. Liu, "Paella: Low-latency model serving with software-defined gpu scheduling," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 595–610.