ATTENTION-BASED MACHINE PERCEPTION FOR INTELLIGENT CYBER-PHYSICAL SYSTEMS

BY

SHENGZHONG LIU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Doctoral Committee:

Professor Tarek F. Abdelzaher, Chair
Professor Lui Sha
Professor Indranil Gupta
Dr. Franck Le, IBM Research

ABSTRACT

Cyber-physical systems (CPS) fundamentally change the way of how information systems interact with the physical world. They integrate the sensing, computing, and communication capabilities on heterogeneous platforms and infrastructures. Efficient and effective perception of the environment lays the foundation of proper operations in other CPS components (*e.g.*, planning and control). Recent advances in artificial intelligence (AI) have unprecedentedly changed the way of how cyber systems extract knowledge from the collected sensing data, and understand the physical surroundings. This novel data-to-knowledge transformation capability pushes a wide spectrum of recognition tasks (*e.g.*, visual object detection, speech recognition, and sensor-based human activity recognition) to a higher level, and opens an new era of *intelligent cyber-physical systems*. However, the state-of-the-art neural perception models are typically computation-intensive and sensitive to data noises, which induce significant challenges when they are deployed on resources-limited embedded platforms.

This dissertation works on optimizing both the efficiency and efficacy of deep-neural-network (DNN)-based machine perception in intelligent cyber-physical systems. We extensively exploit and apply the design philosophy of *attention*, originated from cognitive psychology field, from multiple perspectives of machine perception. It generally means allocating different degrees of concentration to different perceived stimuli. Specifically, we address the following five research questions: First, can we run the computation-intensive neural perception models in real-time by only looking at (*i.e.*, scheduling) the important parts of the perceived scenes, with the cueing from an external sensor? Second, can we eliminate the dependency on the external cueing and make the scheduling framework a self-cueing system? Third, how to distribute the workloads among cameras in a distributed (visual) perception system, where multiple cameras can observe the same parts of the environment? Fourth, how to optimize the achieved perception quality when sensing data from heterogeneous locations and sensor types are collected and utilized? Fifth, how to handle sensor failures in a distributed sensing system, when the deployed neural perception models are sensitive to missing data?

We formulate the above problems, and introduce corresponding attention-based solutions for each, to construct the fundamental building blocks for envisioning an attention-based machine perception system in intelligent CPS with both efficiency and efficacy guarantees.

To Yifei, my parents, and the family, for their dedicated love and support.

ACKNOWLEDGMENTS

At start, I would like to express my deepest gratitude to my advisor, Professor Tarek F. Abdelzaher, for his invaluable guidance, continuous support, and patience through the course of my Ph.D. study. I am always inspired by his passion, vision, and critical thinking for research. His immense knowledge and experience have encouraged me to explore the directions that I am truly interested in. Without his generous support, it would be impossible for me to finish this thesis.

It is also a genuine pleasure to express my deep appreciation to Professor Lui Sha, Professor Indranil Gupta, Doctor Franck Le, for sharing constructive criticism and insightful advices on my Ph.D. research. It is my greatest honor to have them on my Ph.D. thesis committee, collaborate with them on research projects, coauthor research papers, throughout the years. Their insightful feedbacks pushed me to sharpen my thinking and brought my research work to a higher level.

Besides, I would like to take this opportunity to acknowledge my great collaborators over the past years. Special thanks to Dr. Franck Le and Dr. Supriyo Chakraborty for hosting my internship at IBM Research. I also would like to extend my sincere gratitude to Dr. Shuochao Yao, Dr. Huajie Shao, Xinzhe Fu, Dr. Yiran Zhao, Prof. Jiawei Han, Prof. Klara Nahrstedt, Prof. Heechul Yun, Dr. Maggie Wigness, Dr. Philip David, Prof. Mani Srivastava, Prof. Benjamin Marlin, Prof. Archan Misra, Dr. Shaohan Hu, Dr. Shen Li, Yifan Hao, Tai-Sheng Cheng, Prof. Fan Wu, Prof. Zhenzhe Zheng, Jiyang Chen, Ankur Sarker, for their invaluable discussions and helpful advice on my research.

I am always proud and fortunate to be part of the Cyber-Physical Computing Group at University of Illinois at Urbana-Champaign (UIUC). I had great pleasure working with Dongxin Liu, Tianshi Wang, Jinyang Li, Ruijie Wang, Dachun Sun, Yigong Hu, Jinning Li, MD Iftekharul Islam Sakib, Christina Youn, Chaoqi Yang, Zhe Yang, Bo Chen, Hongpeng Guo. Thanks for the pleasure time and hardworking days. Many thanks also to Haotian Wang, Shilei Tian, Dr. Tong Meng, Rui Yang, Jiyong Yu, Zirui Zhao, Xiang Cui, for sharing the happiness and sadness together.

In addition, I would like to thank the U.S. Army Research Labs, Defense Advanced Research Projects Agency, and National Science Foundation for their financial support.

Lastly, I am deeply indebted to my fiancée Yifei Huang, my parents Baoxiang Xu and Weiyao Liu, my sister Meng Liu, as well as Rui Chang, Bin Huang, for their selfless love, comprehension, and support. Therefore, this dissertation is dedicated to them.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Cyber-physical systems (CPS) emerges as a new paradigm defining how information systems (*i.e.*, cyber-side) interact with the physical environment (*i.e.*, physical-side). They integrate the sensing, computing, and communication capabilities on heterogeneous devices and infrastructures, enabling a revolution of "smart era", ranging from smart devices, smart vehicles, to smart homes and smart cities. Benefiting from the transition to autonomy led by CPS, humans are gradually freed from laborious and dangerous works, while the safety and sustainability of the operations are significantly improved at the same time. Representative CPS application fields include autonomous driving, urban surveillance, medical service, agriculture, and disaster rescue.

In these scenarios, obtaining high-quality perception of the physical environment in real-time is the foundation of successful operation of the cyber-physical systems. For instance, in autonomous driving, localizing and categorizing all appeared objects (*e.g.*, lanes, vehicles, pedestrians, traffic lights and signs) surrounding the ego-vehicle in real-time, with camera or LiDAR input, is the basis for subsequent mapping, planning, and control functions to produce timely response. Otherwise, without proper perception, it seems like walking in the dark. In the past decade, a large amount of research efforts have been focused on integrating more sensors into mobile and embedded devices, so that all kinds of sensing applications can be built on top. Nowadays, smart platforms are already equipped with an abundant types of sensors, including cameras, inertial sensors, microphones, light sensors, *et al.*. They significantly improve the sensing ability of these platforms, from the hardware perspective, to perceive the physical environment. Large volume of sensing data is collected by the sensors at every second of operation.

At meanwhile, recent advances in artificial intelligence (AI), especially the application of deep neural networks (DNN), have revolutionized the way of modeling, processing, and learning from various types of sensing data (*e.g.*, video, acoustic signals, inertial sensors, RF sensors) to acquire relevant information and further convert it into knowledge, about physical surroundings. They push a wide spectrum of recognition tasks (*e.g.*, visual object detection, speech recognition, IMU-based human activity recognition) to a higher level, and substantially enhance the perception ability of machines from the learning perspective. This powerful data-to-knowledge transformation capability, successfully bridges the gap between the massive sensing data and the limited modeling capacity of conventional analytics, triggers new generations of complex recognition tasks, and brings us into the new epoch of *intelligent cyber-physical systems*, where deep neural network models are extensively utilized to learn

from the sensing data.

However, it is still a non-trivial task to deploy the deep neural models in practical cyber-physical systems, for two main reasons. First, the state-of-the-art DNNs are computationally intensive and mostly deployed on the powerful cloud in the past, thus induce significant challenges to the limited computing resources on the embedded platforms. Efficiency is critical in machine perception, because the systems rely on the perceived information to carry out proper actions in response to the environment dynamics. Second, the efficacy (*i.e.*, accuracy) of DNNs is challenged by the noisy and unreliable sensing data in CPS. Although DNNs are equipped with powerful modeling capacity on data distributions, they are known to be sensitive to data noises due to their extremely complicated decision boundaries. Noisy data is prevalent in CPS, which can be resulted from multiple perspectives, including heterogeneous sensor manufacture and calibrations, human interventions, unstable network connections, and environment diversity. DNNs need to be specially adapted or augmented to handle these practical data imperfections during the deployment.

My dissertation works on optimizing both *efficiency* and *efficacy* of DNN-based perception systems in intelligent CPS, under limited computing resources and unreliable sensing data. The central design philosophy is inspired by the concept of *attention* [1] in cognitive psychology, which originally refers to the cognitive behavior that human brains can only selectively concentrate on one discrete stimulus among multiple perceivable stimuli[1]. For example, when reading a book, our visual attention can only be focused on a specific area of the page, *i.e.*, the few lines that we are currently reading, although we perceived many lines as visual stimuli. Attention essentially represents an active resource allocation mechanism that processes different parts of the input with different degrees of importance, with limited cognitive processing resources. We interpret attention from two different perspectives when applying the idea to machine perception, in terms of efficiency and efficacy, respectively. From the efficiency perspective, consider the bottleneck computation resource (*i.e.*, GPU) as an analogy of our brain that can only process limited amount of data at every second. In order to achieve real-time (visual) machine perception on such resource-limited platforms, we only want the expensive neural networks to focus on (*i.e.*, process) "the most important region(s)" of the input and ignore the remaining regions. This forms a novel real-time scheduling problem for machine perception pipeline. From the efficacy perspective, when making predictions based on sensing data collected from heterogeneous sensing modalities and diverse locations, we want the DNN models to enhance the impact of important parts of the sensing data and filter out the irrelevant noises. This dynamic weighting mechanism

[1] `https://courses.lumenlearning.com/boundless-psychology/chapter/attention/`

is called *neural attention* [2] in machine learning field. It becomes a novel sensor fusion problem within DNN based perception models.

With that central design philosophy explained, we establish the following statement for this dissertation: *By applying the attention-based design philosophy, we build an efficient and effective machine perception pipeline for intelligent cyber-physical systems, that can produce high-quality predictions in real-time on resource-limited computing platforms, using possibly unreliable and noisy sensing data.*

Specifically, we tackle the following research problems towards building attention-based perception pipelines. First, can we partition the input data (*e.g.*, image) into fine-grained and semantically-meaningful partial regions, such that we can correspondingly schedule their processing? If yes, do we need external cueing from another sensor input to assist the data partitioning/slicing, and how to schedule their processing tasks to achieve real-time machine perception without degradation on perception quality? Second, can we eliminate the dependency on the external cueing sensor, and make the system work in a self-cueing manner, by exploiting the temporal correlations in the data stream? Third, how to extend the slicing-and-scheduling pipeline to the distributed sensing systems so that the perception efficiency can be further improved by exploiting the spatial data correlations among the sensors? Fourth, how to optimize the achieved inference accuracy of the DNN perception models, by designing and integrating appropriate neural attention mechanisms for heterogeneous sensor data fusion? Finally, can we extend the dynamic sensor fusion mechanisms to missing sensor scenarios, such that the same DNN model can work properly under different sensor availability (*i.e.*, when only a subset of sensors involved in the training are available)?

In the following sections, we give an overview of the solutions we proposed for each of the above research problems.

1.1 ATTENTION-BASED PERCEPTION SCHEDULING WITH EXTERNAL-CUEING

This work first discusses *algorithmic priority inversion* in mission-critical machine inference pipelines used in modern neural-network-based cyber-physical systems, and develops a scheduling solution to mitigate its effect. In general, *priority inversion* occurs in real-time systems when computations that are of lower priority are performed together with or ahead of those that are of higher priority. Technically, there are two competing requirements that inform task prioritization, namely, *criticality* and *urgency*. Inversion occurs if priorities derived from these requirements are not obeyed. In current machine intelligence software, significant priority inversion occurs on the path from perception to decision-making, where the execution of underlying neural network algorithms does not differentiate between crit-

ical and less critical data. We design a scheduling framework to resolve this problem, and demonstrate that it improves the system's ability to react to critical inputs, while reducing the platform cost at meanwhile. The framework is prototyped in visual perception systems of autonomous vehicles, where the input from a ranging sensor, *i.e.*, LiDAR, is used as the external cue to guide the slicing of camera images, then the sliced images are fed into the scheduling algorithm as input. As an extension, we evaluate two specific criticality designs, namely distance-based criticality and relative velocity-based criticality, about their impact on the achieved prioritization mechanism. In these cases, the allocation of attention is directly associated with the designated physical measures, *i.e.*, distance or relative velocity, which is similar to how humans process the perceived visual information while driving. As another extension, we explore the further improvement brought by an image resizing mechanism. The details of this work are described in Chapter 2.

1.2 ATTENTION-BASED PERCEPTION SCHEDULING WITH SELF-CUEING

This work extends the previous work by eliminating the dependency on external cueing sensors, and presenting a self-cueing real-time framework for attention scheduling in AI-enabled visual perception systems that minimizes a notion of system uncertainty. By *attention scheduling* we refer to the challenge of having the visual perception subsystem (neural network) inspect parts of the scene before others in some criticality-aware fashion. By *self-cueing*, we refer to an architecture where attention prioritization does not require the use of external cueing sensors, thereby simplifying design and avoiding the need for sensor synchronization. The objective is to enable real-time tracking of objects detected using vision-based neural network models on resource-limited embedded platforms. Instead of performing object detection on every full-scale camera frame, we take advantage of the temporal correlations in video streams and only run the full-frame detections intermittently. Between them, a set of object-oriented, criticality-based, and uncertainty-aware *partial-frame detections* are scheduled to track regions of interest. The system relies on two components: First, an optical flow-based object tracking and data slicing module that uses previously detected object locations, and pixel motions estimated between pairs of neighboring frames, to predict future object trajectories. Second, a novel scheduling policy that decides when to execute the partial-frame detection tasks for each tracked object, in order to minimize the maximum uncertainty on their locations over time. In addition, a task batching mechanism is integrated into the scheduling algorithm to optimize the task processing capacity on modern GPUs. We evaluate the proposed scheduling architecture through extensive evaluations with real-word driving datasets on NVIDIA Jetson Xavier. The details of this work are

described in Chapter 3.

1.3 ATTENTION-BASED MULTI-CAMERA SCHEDULING

This paper further extends the previous two works from single-camera perception to multi-camera perception scenarios. It presents a real-time multi-view scheduling framework for AI-enabled live video analytics at the edge. We regard object detection and tracking as the scheduling workload. We consider scenarios where multiple cameras are deployed around a local area (*e.g.*, a traffic intersection) such that one object may be observable from multiple views. Each camera has access to limited onboard computing capacity. In order to perform efficient real-time detection and tracking, we exploit the spatial-temporal correlations among multi-camera video streams. First, to exploit the *temporal correlations* in video streams, we only run object detection on full camera frames with limited intermittency, and a set of object-based partial frame detections are scheduled in-between. Second, to further benefit from the *spatial view overlaps across cameras*, we only schedule one camera to track an object if the object is observable by multiple cameras. Specifically, we use a data-driven object correlation model to identify common objects across cameras, and then propose a batch-aware load-balanced (BALB) scheduling algorithm to calculate the object-to-camera assignment, which specifies the subset of objects to be tracked at each camera. We evaluate the proposed system on a testbed consisting of multiple NVIDIA Jetson platforms with heterogeneous hardware configurations. The details of this work are described in Chapter 4.

1.4 ATTENTION-BASED MULTI-SENSOR FUSION

This paper enhances deep-neural-network-based inference in sensing applications by introducing a lightweight attention mechanism called the *global attention module* for multi-sensor information fusion. This mechanism is capable of utilizing information collected from higher layers of the neural network to selectively amplify the influence of informative features and filter unrelated noise at the fusion layer. We successfully integrate this mechanism into a new end-to-end learning framework, called *GlobalFusion*, where two global attention modules are deployed for spatial fusion and sensing modality fusion, respectively. We evaluate the effectiveness of GlobalFusion at improving information fusion on multi-sensor human activity recognition (HAR) tasks. In addition, we observe the learned attention weights agree well with human intuition, from the interpretability perspective. Only a negligible overhead is induced by the global attention modules, in terms of both inference time and

energy consumption, when the models are deployed on commodity IoT devices. The details of this work are described in Chapter 5.

1.5 ATTENTION-BASED MISSING SENSOR RECOVERY

This paper enhances the robustness of neural perception models to the missing sensor problem, when only a subset of sensors used for training are available during the inference, by introducing a novel feature reconstruction module, namely the *graph recovery* module, that handles missing sensors directly inside the neural network, when there is sufficient redundancy in the sensor coverage. Specifically, we consider topology-aware multi-nodes sensing systems, where sensors are placed on a physically interconnected network. We design a novel neural message passing mechanism that logically mimics physical network topology, based on recent advances in graph neural networks (GNN). We rely on a spatial locality assumption, where the data correlations between physically connected sensors are explicitly explored. When encountering missing sensors, information is passed from available sensors to missing sensors to be used to reconstruct their features. Moreover, at each message passing step, we utilize a gating mechanism inspired by Gated Recurrent Units (GRU) to automatically control the information flow between available sensors and missing sensors. We extensively evaluate the reconstruction performance of the graph recovery module with two representative IoT applications, human activity recognition (HAR) and electroencephalogram (EEG)-based motor-imagery classification. The details of this work are described in Chapter 6.

1.6 DISSERTATION ORGANIZATION

The rest of this dissertation is organized as follows: Part 1 (*i.e.*, Chapter 2-4) introduces the works on attention-based efficiency optimization for machine perception. In Chapter 2, we introduce the cross-cueing scheduling framework for criticality-aware vision perception tasks, that depends on an external sensor to slice the input image input into semantically meaningful partial regions. As extensions, we not only instantiate two different criticality designs, but also explore the further improvement brought by image resizing mechanisms. In Chapter 3, we improve the generalizability of visual perception scheduling, by eliminating the dependency on external cueing sensors. Instead, a self-cueing framework that purely bases on the temporal correlations in video streams to slice the images, is introduced. A novel uncertainty-driven, batch-aware scheduling algorithm is proposed to minimize the notion of

maximum uncertainty on object locations. In Chapter 4, we further extend the scheduling framework to distributed multi-camera perception systems, where both spatial and temporal correlations within distributed video streams are leveraged. Part 2 (*i.e.*, Chapter 5-6) applies attention-based designs to optimize the efficacy of machine perception models. In Chapter 5, we propose a novel *global attention* module that uses high-level neural network features, that reside close to the output layer, to selectively enhance the impact of relevant sensors/locations, and ignore irrelevant noises, during low-level sensor fusion. In Chapter 6, we address how to handle missing sensor issues within neural perception models by relying on information on available sensors to reconstruct the features for missing sensors. Finally, we conclude the dissertation and discuss some future works in Chapter 7.

CHAPTER 2: ATTENTION-BASED SCHEDULING WITH EXTERNAL-CUEING

2.1 OVERVIEW

This chapter introduces the notion of *algorithmic priority inversion* that plagues modern *mission-critical* machine inference pipelines. We describe an initial solution towards removing such priority inversion from neural-network-based systems to support real-time intelligent cyber-physical applications. As a running application, we consider autonomous driving, although we expect the design principles described in this work to remain applicable in other contexts.

Importantly, to set the scope, we distinguish between *safety-critical* and *mission-critical* design requirements of cyber-physical systems. A safety-critical requirement might be to guarantee collision-avoidance. A mission-critical requirement might be to do valid path planning, taking into account anticipated future mobility of other agents. On the surface there may appear to be overlap; the computed path must still avoid running into other objects. The distinguishing property is that the *mission-critical subsystem has lower reliability requirements*. Hence, it may speculatively use less certain data (e.g., an anticipated future trajectory of neighboring objects), and is allowed to occasionally err. The safety-critical subsystem should be able to override and restore safety when needed. For example, if the mission-critical subsystem mispredicts another object's future path, leading to a possible collision, the safety-critical subsystem should eventually detect the imminent threat and perform emergency collision-avoidance.

The application of artificial intelligence to cyber-physical systems poses different challenges in the different subsystems. One challenge is to continue to meet *safety-critical* design requirements by the safety-critical subsystem. General machine learning solutions that offer such strong safety assurances are notoriously difficult in practice and are out of scope for this work. In fact, for the purposes of this work, we can imagine the safety-critical subsystem to be AI-free (e.g., reliable ranging sensors that detect dangerous proximity of other objects and invoke emergency intervention).

This work, instead, focuses on the *mission-critical* subsystem. The challenge addressed is to optimize the schedulability of mission-critical real-time perception tasks in this subsystem to remove priority inversion. Perception is one of the key components that enable system autonomy. It is also a major efficiency bottleneck that accounts for a considerable fraction of resource consumption [3, 4]. In general, priority inversion occurs in real-time systems when computations that are less critical (or that have longer deadlines) are performed together

with or ahead of those that are more critical (or that have shorter deadlines). Current neural-network-based machine intelligence software suffers from a significant form of priority inversion on the path from perception to decision-making, because current algorithms process input data sequentially, as opposed to processing important parts of a scene first. This limitation may result in inferior system responsiveness to critical events, or (equivalently) increased cost of hardware to meet mission needs. By resolving this problem, we significantly improve system ability to react to critical inputs at a lower platform cost. The work applies to intelligent cyber-physical systems that perceive their environment in real time (using neural networks), such as self-driving vehicles [5], autonomous delivery drones [6], military defense systems [7], and socially-assistive robotics [8].

To understand the present gap, observe that current perception-related neural networks perform many layers of manipulation of large multidimensional matrices (called *tensors*). Yet, the current state of the art in designing the underlying neural network libraries (e.g., *TensorFlow*) is reminiscent of what used to be called the *cyclic executive* [9] in early operating system literature. Cyclic executives, in contrast to priority-based real-time scheduling [10], processed all pieces of incoming computation at the same priority and quality (e.g., as nested loops). Similarly, given incoming data frames (e.g., multi-color images or 3D LiDAR point clouds), modern neural network algorithms process all data rows and columns at the same priority and quality, with no regard to cues from the physical environment that impact time-constraints and criticality of different parts of the data scene.

This flat processing is in sharp contrast to the way *humans* process information. Humans have an innate ability to not only perceive their environment, but also make critical and timely attention allocation decisions that help us expend limited cognitive resources where they are most needed in a critical dynamic situation. For example, given a complex scene, such as a freeway where one of the nearby vehicles appears to have temporarily lost control of steering, human drivers are good at understanding what to focus on to plan a valid path forward amidst the resulting confusion.[1] This capability is substantially different from, say, attention mechanisms used in machine inference [11, 12], where attention is related to logical computational weights assigned to different inputs as opposed to prioritized allocation of actual processing resources. Different from the neural attention mechanisms [2, 13, 14, 15, 16] that automatically computes the logical weights to optimize the model performance (*i.e.*, efficacy optimization), our proposed attention mechanism injects extra human supervision

[1]Note that, by planning a path that continues to make forward progress, we are talking about a mission-critical function (assuming the mission involves making progress towards a destination). In contrast, a safety-critical override might simply stop the vehicle to avoid a collision. Clearly stopping the vehicle will stop progress towards mission objectives, but may ensure safety.

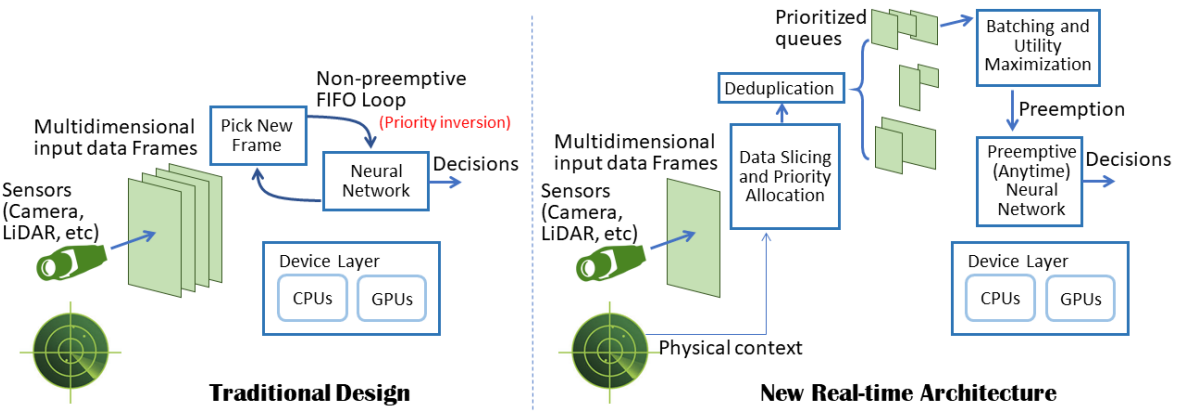


Figure 2.1: Real-time Machine Inference Pipeline Architecture.

(*i.e.*, the criticality design) into the resource allocation to prioritize the "critical parts" under our definition with physical correspondence (*i.e.*, efficiency optimization).

The lack of prioritized allocation of processing resources to different parts of an input data stream (e.g., from a camera) creates what we henceforth call *algorithmic priority inversion*. In the above example, all pixels of the entire freeway scene are processed by the same algorithm at the same priority, as opposed to giving the runaway vehicle more attention while possibly temporarily ignoring other less important elements of the scene (e.g., far-away objects).

We develop an architecture for separating input data (to be processed by the neural-network) into regions of different criticality, and assigning different deadline-driven priorities to the processing of these regions. We then introduce a utility-optimizing scheduling algorithm for the resulting real-time workload to meet deadlines while maximizing a notion of global utility (to the mission). We implement the architecture on an NVIDIA Jetson AGX Xavier platform, and do a performance evaluation on the platform using real video traces collected from autonomous vehicles. The results show that the new algorithms significantly improve the average quality of machine inference, while nearly eliminating deadline misses, compared a set of state-of-the-art baselines executed on the same hardware under the same frame rate.

2.2 SYSTEM ARCHITECTURE

Consider an intelligent cyber-physical system equipped with a camera that observes its physical environment, a neural network that processes the observations, and a control unit that must react in real time. As mentioned earlier, we focus on scheduling of perception tasks in the mission-critical subsystem. For example, the neural network might identify the

types of objects present in the field of view so that subsequent path planning can be done accordingly. Figure 2.1 contrasts the traditional design of machine inference pipelines in such systems to the proposed architecture. In the traditional design, input data frames captured by sensors are processed sequentially by the neural network. Network execution is typically non-preemptive. It considers one frame at a time, producing an output on each frame before the next frame is handled.

Unfortunately, the multi-dimensional data frames captured by modern sensors (e.g., colored camera images and 3D LiDAR point clouds) carry information of different degrees of criticality in every frame.[2] Data of different degrees of criticality may require a different processing latency. For example, processing parts of the image that represent far away objects does not need to happen every frame, whereas processing nearby objects, such as a vehicle in front, needs to be done immediately because the nature of nearby objects (e.g., car versus pedestrian) has impact on immediate path planning. To accommodate these differences in input data criticality, we propose a novel mission-critical subsystem architecture that breaks the path from perception to decision-making into four components:

- *The data slicing and priority allocation module:* This module breaks up newly arriving frames into smaller regions of different degrees of criticalty based on simple heuristics (e.g., closer objects need to be attended to first).

- *The deduplication module:* This module drops redundant regions (i.e., ones that refer to the same physical objects) across successive arriving frames.

- *The "anytime" neural network:* This neural network implements an imprecise computation model that allows execution to be preempted, while yielding partial utility from the partially completed computation. The approach allows newly arriving critical data to preempt the processing of less critical data from older frames.

- *The batching and utility maximization module:* This module sits between the data slicing and deduplication modules on one end and the neural network on the other. With data regions broken by priority, it decides which regions to pass to the neural network for processing. Since multiple regions may be queued for processing, it also decides how best to benefit from batching (that improves processing efficiency). A utility maximizing algorithm controls the produced schedule to maximize a quality metric.

[2]By different degrees of *criticality*, we are referring to different levels of importance within the *mission-critical* subsystem. For example, far-away objects are less relevant to path planning than nearby objects. We are not refering to a distinction between safety-critical and mission-critical data.

Since our purpose is to mitigate priority inversion on the path from *perception* to decision-making, we shall refer to the subsystem shown in Figure 2.1 as the *observer*. The goal is to allow the observer to respond to more urgent stimuli ahead of less urgent ones. The main contribution of this paper lies in the design of the *batching and utility maximization module* that maximizes the quality of inference while meeting response deadlines. For completeness, below we first describe all of the above components of the observer, respectively. We then detail the batching and utility maximization algorithm used.

2.2.1 Data Slicing and Priority Allocation

This module breaks up input data frames into regions that require different degrees of attention. Objects with a smaller *time-to-collision* [17] should receive attention more urgently. We further assume that the observer is equipped with a *ranging* sensor. For example, in autonomous driving systems, a LiDAR sensor measures distances between the vehicle and other objects. LiDAR point cloud based object localization techniques have been proposed in recent literature [18]. They provide a fast (i.e., over 200 Hz) and accurate ranging and object localization capability. The computed object locations can then be projected onto the image obtained from the camera, allowing the extraction of regions (subareas of the image) that represent these localized objects, sorted by distance from the observer. In this chapter, we assume that errors in LiDAR-based slicing are negligible[3]. The extraction of such subareas is the main function of the data slicing module. In this paper, for simplicity, we restrict those subareas to rectangular regions. We call them *bounding boxes*. The other function of the module is prioritization (of bounding boxes) by time-to-collision, given the trajectory of the observer and the location of the object. Computing the time-to-collision is a well-studied topic and is not our contribution [17]. We list it as one of our future directions to integrate more complex and practical object priority designs into our framework.

2.2.2 Deduplication

The function of the deduplication module is very simple. It elimiates redundant bounding boxes. Since the same objects will generally persist across many LiDAR and camera frames, the same bounding boxes will be identified in multiple frames. The set of bounding boxes pertaining to the same object in different frames is called a *tubelet*. In real-time systems, in general, the best information is the most recent. Thus, only the most recent bounding box in a tubelet needs to be acted on. Boxes with significant location overlap from frame

[3]Perfect sensor-based cueing is hard, human attentions could be misplaced from time to time

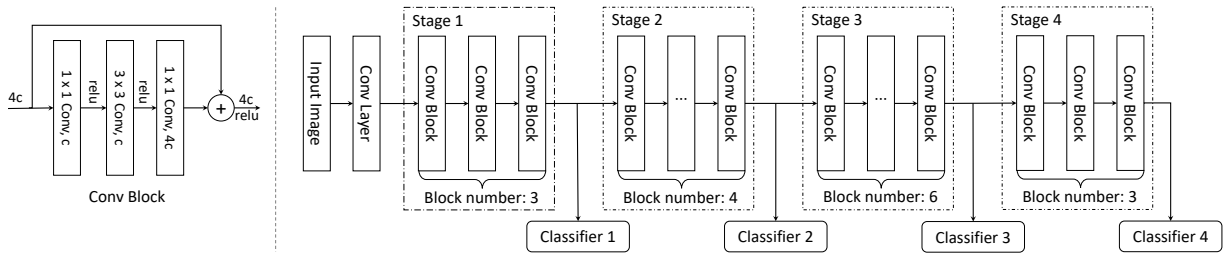


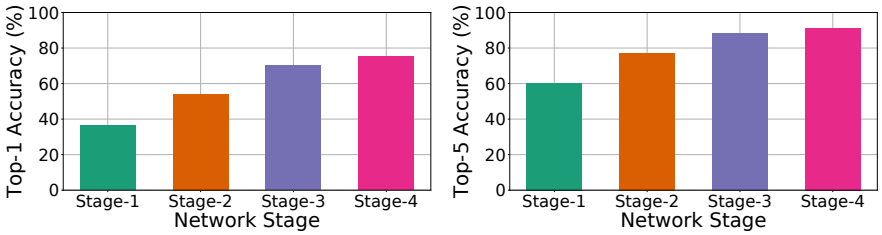
Figure 2.2: ResNet [19] architecture with 4 stages and 50 layers. In the left part, we show the design of bottleneck block, which is the basic building block of ResNet. c represents the feature dimension. The classifier is simply the concatenation of a max pooling layer and a fully connected layer.

to frame are considered redundant. The deduplication module identifies boxes with large overlap and stores the most recent box only. For efficiency reasons described later, we quantize the used bounding box sizes. The deduplication module uses the same box size for the same object throughout the entire tubelet. If the underlying object changes location enough for the bounding box to jump to another size category, the overlap between the two boxes (of different size) will be small enough that the module will fail to recognize them as the same object. This creates a minor loss of deduplication efficiency but simplifies the forecasting of execution time (used by the scheduler) associated with processing what the module recognizes as the same object (since the size of its bounding box does not change).

Note that, in a traditional neural network processing pipeline, each frame is processed in its entirety before the next one arrives. Thus, no deduplication module is used. The option to add this time-saving module in our architecture arises because our pipeline can postpone processing of some objects until a later time. By that time, updated images of the same object may arrive. This enables savings by looking at the latest image only, when the neural network eventually gets around to processing the object.

2.2.3 The Anytime Neural Network

A perfect *anytime* algorithm is one that can be terminated at any point, yielding utility that monotonically increases with the amount of processing performed. Our neural network approximates that model. Specifically, it implements an *imprecise computation* model [20, 21, 22] that provides usable and approximate partial results. In an imprecise computation model, processing consists of two parts: a *mandatory part* and an *optional part*. The optional part, or a portion thereof, can be skipped to conserve resources. When the optional part is skipped, the task is called to produce an *imprecise* (i.e., approximate) result. In imprecise computation models, the processing fidelity improves as we execute more stages.



(a) Top-1 Accuracy (b) Top-5 Accuracy

Figure 2.3: ResNet stage accuracy change on ImageNet [24] dataset.

Deep neural networks (e.g., image recognition models [19]) are a concatenation of a large number of layers that can be divided into several stages, as we show in Figure 2.2. Ordinarily, an output layer is used at the end to convert features computed by earlier layers into the output value (e.g., an object classification). Prior work has shown, however, that other output layers can be forked off of intermediate stages producing meaningful albeit imprecise outputs based on features computed up to that point [23]. Figure 2.3 shows the accuracy of ResNet-based classification applied to the ImageNet [24] dataset at intermediate stages of neural network processing. It shows that neural network inference can be divided into a mandatory part and optional parts. The quality of outputs increases when the network executes more optional parts. Thus, network execution can be aborted (e.g., in favor of a new more important task) short of executing all its optional parts, yielding partial utility as described in recent work [23].

An essential component in this model is the choice of task utility, which lays the foundation for time assignment to the processing of different objects/tasks. In this paper, we set utility (from the task's output) proportionally to *confidence in result*; a low confidence output is less useful than a high confidence output. The proportionality factor itself can be set depending on task criticality, such that uncertainty in output of more critical tasks is penalized more. We adopt the algorithm proposed by Yao *et al.* in RDeepSense [25] to estimate expected confidence in outputs of future neural network stages. This allows us to compute the expected utility of each stage before it is executed.

There are two existing limitations associated with the imprecise computation model for neural networks. First, it currently is only applicable to classification networks, but not object detection networks. Thus, our approach here relies completely on LiDAR slicing to localize the object locations. Second, there is a tradeoff between the model accuracy on different stages where only a suboptimal result is achieved at each stage. These two aspects inspire us to explore an alternative approach using image resizing to define the degrees of allocated resources in a following extension that can overcome both limitations by design.

2.2.4 Batching and Utility Maximization

This module decides the schedule of processing of all regions identified by the data slicing and prioritization module and that pass de-duplication. As discussed above, utilizing LiDAR point clouds to efficiently localize objects in each frame [18, 26], the data slicing module computes *bounding boxes* for objects detected. These boxes constitute regions that require attention, each assigned a degree of criticality. The deduplication module groups boxes related to the same object into a tubelet. Only the latest box in the tubelet is kept. The remaining boxes need not be processed. Each physical object gives rise to a separate neural network task to be scheduled. The input of that task is the bounding box for the corresponding object (cropped from the full scene). Below, we describe how the batching and utility maximization module schedules the tasks that process the different bounding boxes.

2.3 THE SCHEDULING PROBLEM

In this section, we describe our task execution model and formulate the scheduling problem studied in this work. We then derive a near-optimal solution.

2.3.1 The Execution Model

As alluded to earlier, the scheduled tasks in our system constitute the execution of multi-layer deep neural networks (*e.g.*, ResNet [19], as shown in Figure 2.2), each processing a different input data region (a bounding box). As shown in Figure 2.2, tasks are broken into stages. Each stage includes multiple neural network layers. The unit of scheduling is a single stage. Neural network execution of a single stage is non-preemptive, but tasks can be preempted on stage boundaries. A task arrives when a new object is detected by the ranging sensor (*e.g.*, LiDAR) giving rise to a corresponding new bounding box in the camera scene. Let the arrival time of task τ_i be denoted by a_i. A deadline $d_i > a_i$, is assigned by the data slicing and priority assignment module denoting the time by which the task must be processed (*e.g.*, the corresponding object classified). The data slicing and priority assignment module is invoked at frame arrival time. Therefore, both a_i and d_i are a multiple of frame inter-arrival time, H. Since new objects can appear in the field of view at any time, we do not pose periodicity assumptions on object arrival times and deadlines. No task can be executed after its deadline. Future object sizes, arrival times, and deadlines are unknown, which makes the scheduling problem an *online decision problem*. A combination

Table 2.1: Table of Notations.

Symbol	Meaning
H	Camera sampling period.
h	Time index within a period.
δ	Minimum time unit. All times are multiples of δ.
i	Object index.
j	Neural network stage index.
P	Batch of tasks.
$\mathcal{S}$	Available image size set, $\lvert\mathcal{S}\rvert = K$.
k	Image size index, which refers to the k-th image size in $\mathcal{S}$.
b	Batch size.
t	Index for scheduling period, counted in multiples of H.
τ_i	The task related to the i-th object.
s_i	Image size for the i-th object.
l_i	Number of execution stages for the i-th object.
a_i, d_i	Arrival time and deadline for the i-th object.
$e_{j,b}^{(k)}$	Exec. time of stage j when batching b images of size k.
$L^{(k)}$	Neural network stage number for image size k.
L_i	Available neural network stage number of tasks τ_i.
L	Max stage number among all image sizes.
E	Ratio between the max stage time and the min stage time.
$B^{(k)}$	Batching constraint of image size k.
B	Max batch size among all image sizes.
$R_{i,j}$	Aggregated utility for executing the i-th task for j stages.
$\mathcal{T}(t)$	Available task set at t-th scheduling period.

of two aspects make this real-time scheduling problem interesting:

Batching: Stages of the neural networks are executed on a GPU. We are particularly interested in lower-priced GPUs. While such GPUs feature parallel execution, one way of exploiting their computation capabilities is to execute the same kernel on all GPU cores. This means that we can run *different* tasks concurrently on the GPU as long as we run the *same kernel* on all GPU cores. We call the assembly of such concurrently executable task sets, *batching*. Running the same kernel on all GPU cores means that we can only batch tasks if both of the following applies: (i) they are executing *the same neural network stage* and (ii) they *run on the same size inputs*. The latter condition is because the processing of different bounding box sizes requires instantiating different GPU kernels. Batching that satisfies the above two conditions ensures that the same kernel is executed on all cores. Batching is advantageous because it allows us to better utilitize the GPU, so we want to take advantage of it in scheduling. To increase batching opportunities, we limit the size of possible bounding boxes used by the data slicing module to a finite set of of options. For a given bounding box size k, at most $B^{(k)}$ tasks (processing inputs) can be batched together before overloading GPU capacity. We call it the *batching limit* for the corresponding input size.

Imprecise Computations: Let the number of stages in the neural network for task τ_i be denoted by L_i (normally, this is the same number of all tasks, but it may depend on the size on the input object). We call the first stage *mandatory* and call the remaining stages *optional*. Following a recently developed neural network implementation as imprecise computations [27], tasks are written such that they can return an object classification result once the mandatory stage is executed. This result then improves with the execution of each optional stage. Earlier work presented an approach to estimate the expected confidence in correctness of results of future stages, ahead of executing these stages [25]. This estimation offers a basis for assessing utility of future task stage execution. We denote the utility of task τ_i after executing $j \leq L_i$ stages by $R_{i,j}$, where $R_{i,j}$ is set proportionately to the predicted confidence in correctness at the conclusion of stage j, computed as proposed by Yao *et al.* [25]. Note that, the expected utility can be different among tasks (depending in part on input size), but it is computable, non-decreasing, and concave with respect to the network stage [25].

We denote by $\mathcal{T}(t)$ the set of *current tasks* at scheduling period t. A task, τ_i, is called *current* at period t, if $a_i \leq t < d_i$, and the task has not yet completed its last stage, L_i. For task τ_i of input size, k, the execution time of the j-th stage is denoted by $e_{j,b}^{(k)}$, where b is the number of tasks that are batched together during the execution of that stage. Since batched tasks execute concurrently, in principle, $e_{j,b}^{(k)}$ should not depend on b. In reality, however, there is a data copying cost in and out of the GPU that depends on the total number of batched tasks, leading to a slight increase in concurrent execution time with batching. Later in the evaluation section, we profile this effect. With the above description of our execution model, we are now ready to formulate the new scheduling problem, which we call the *Batched Online Object-recognition Scheduling with Imprecise Computation (BOOSIC)* problem, below.

2.3.2 Problem Formulation

The problem addressed in this work is simply to decide on the number of stages $l_i \leq L_i$ to execute for each task τ_i and to schedule the batched execution of those task stages on the GPU such that the total utility, $\sum_i R_{i,l_i}$, of executed tasks is maximized, and batching constraints are met (*i.e.*, all used GPU cores execute the same kernel at any given time, and that the batching limit is not exceeded). While the deadlines do not appear as explicit constraints in this formulation, the deadline miss ratio can be made arbitrarily small by associating deadline misses with an arbitrary large negative utility. Equivalently, one can raise the utility of timely stage execution by the same offset (and set the utility from missing

a deadline to zero). In summary:

The BOOSIC problem: *With online task arrivals, the objective of the BOOSIC problem is to derive a schedule x to maximize the aggregate system utility. The schedule decides three outputs: task stage execution order on the GPU, task execution depth (*i.e.*, number of stages to execute of each task), and task batching (which tasks to execute together). Specifically, for each scheduling period t, we use $x_t(i,j) \in \{0,1\}$ as an indicator variable to denote whether the j-th stage of task τ_i is executed. Besides, we use P to denote a batch of tasks, where $|P|$ denotes the number of tasks being batched. The mathematical formulation of the optimization problem is:*

$$BOOSIC: \max_{x} \sum_{t} \sum_{i} x_t(i,j)\,(R_{i,j} - R_{i,j-1}) \tag{2.1}$$

$$\text{s.t. } x_t(i,j) \in \{0,1\},\ \sum_{t=1}^{T} x_t(i,j) \leq 1, \quad \forall i,j \tag{2.2}$$

$$x_t(i,j) = 0, \quad \forall t \notin [a_i, d_i),\ \forall i,j \tag{2.3}$$

$$\sum_{t'=1}^{t-1} x_{t'}(i,j-1) - x_t(i,j) \geq 0, \quad \forall i,j > 1, t > 1 \tag{2.4}$$

$$s_i = s_{i'} = k,\ l_i = l_{i'},\ |P| \leq b_k, \quad \forall i \in P,\ i' \in P,\ \exists k \in \mathcal{S} \tag{2.5}$$

The following set of constraints have to be satisfied: (2.2) Each network stage for each task can only be executed once; (2.3) No task can be executed after its deadline; (2.4) The execution of different stages of the same task must satisfy their precedence constraints; (2.5) Only tasks with the same (image size, network stage) can be batched, and the number of batched tasks can not exceed the batching constraint of their image size.

Only one batch (kernel) can be executed on the GPU at any time. However, multiple batches can be executed sequentially in one scheduling period, as long as the sum of their execution times does not exceed the period length, H. Next, we present an online scheduling framework for reasoning about our BOOSIC problem, and propose a set of scheduling algorithms that offer different trade-offs between optimality and execution overhead.

2.3.3 An Online Scheduling Framework

We derive an optimal dynamic-programming-based solution for the BOOSIC scheduling problem and express its competitive ratio relative to a clairvoyant scheduler (that has full

knowledge of all future task arrivals). We then derive a more efficient greedy algorithm that approximates the dynamic programming schedule. We define the clairvoyant scheduling problem as follows:

Definition 2.1 (The Clairvoyant Scheduling Problem)**.** Given information of all future tasks that will arrive, the clairvoyant scheduling problem seeks to maximize the aggregate utility obtained from (stages of) tasks that are completed before their deadlines. The maximum aggregate utility is defined as OPT.

With no knowledge of the future, an online scheduling algorithm that achieves a competitive ratio of c (*i.e.*, a utility greater than or equal to $\frac{1}{c} \cdot OPT$) is called c-competitive. A lower bound on the competitive ratio for online scheduling algorithms was shown to be 1.618 [28].

Our scheduler is invoked upon frame arrivals, which is once every H units of time. We thus call H the *scheduling period*. We assume that all task stage execution times are multiples of some basic time unit, thereby allowing us to express H by an integer value (*i.e.*, an integer multiple of the basic time unit δ). We further call the problem of scheduling current tasks within the period between successive frame arrivals, the *local scheduling problem*:

Definition 2.2 (The Local BOOSIC Scheduling Problem)**.** Given the set of current tasks, $\mathcal{T}(t)$, within scheduling period, t, the local BOOSIC scheduling problem seeks to maximize the total utility gained within this scheduling period only.

We proceed to show that an online scheduling algorithm that optimally solves the local scheduling problem within each period will have a good competitive ratio. Let L be the maximum number of stages in any task, and let B be the maximum batching size:

Theorem 2.1. If during each scheduling period, the local BOOSIC scheduling problem for that period is solved optimally, then the resulting online scheduling algorithm is $\min\{2 + L, 2B + 1\}$-competitive (with respect to a clairvoyant algorithm).

Proof. We prove the theorem using charging arguments. Throughout the proof, we will refer to stages of tasks also as "tasks". We define $\{\mathcal{T}^*(t)\}_{t=1,\ldots,T}$ as the set of tasks executed under an optimal Clairvoyant scheduling algorithm during each scheduling period t. Define $\{\mathcal{T}_{alg}(t)\}_{t=1,\ldots,T}$ as the set of tasks executed under an arbitrary online scheduling algorithm that satisfies the condition of Theorem 2.1 during each scheduling period. We will charge the utility of $\{\mathcal{T}^*(t)\}_{t=1,\ldots,T}$ to that of $\{\mathcal{T}_{alg}(t)\}_{t=1,\ldots,T}$ using two schemes. We will show that each task in $\{\mathcal{T}_{alg}(t)\}_{t=1,\ldots,T}$ is charged no more than $2 + L$ times in the first scheme and no more than $2B + 1$ times in the second scheme.

Consider a generic period t. In the first scheme, for each task in $\mathcal{T}^*(t)$, if its first stage is executed by the online scheduling algorithm, then we charge the utility of the task to its first stage in $\{\mathcal{T}_{alg}(t)\}_{t=1,\ldots,T}$. Each task is charged for at most L times in this process. Let $\hat{\mathcal{T}}^*(t) \subseteq \mathcal{T}^*(t)$ be the tasks that are executed under the optimal Clairvoyant scheduling algorithm whose first stage are never executed under the online scheduling algorithm. Let $\hat{\mathcal{T}}_1^*(t)$ be the set of tasks in $\hat{\mathcal{T}}^*(t)$ that completely lie in period t under the schedule of the optimal Clairvoyant algorithm and let $\hat{\mathcal{T}}_2^*(t)$ be the set of tasks in $\hat{\mathcal{T}}^*(t)$ that span period t and $t+1$. By definition $\hat{\mathcal{T}}_1^*(t) \cup \hat{\mathcal{T}}_2^*(t) = \hat{\mathcal{T}}^*(t)$. Since the first stages of tasks in $\hat{\mathcal{T}}_1^*(t)$ and $\hat{\mathcal{T}}_2^*(t)$ are all available at the beginning of period t, by the concavity of the utilities of tasks, we have that the total utility of tasks in $\hat{\mathcal{T}}_1^*(t)$ and the total utility of those in $\hat{\mathcal{T}}_2^*(t)$ are both smaller than or equal to the local optimal utility. Since the considered online scheduling algorithm achieves local optimal, we can charge the utility of $\hat{\mathcal{T}}_1^*(t)$ to that of $\mathcal{T}_{alg}(t)$ with each task in the latter set being charged no more than 1 time. The same can be done for tasks in $\hat{\mathcal{T}}_2^*(t)$. Following this charging scheme for all t, we charge the utilities of all the tasks executed under the offline optimal to that of the online algorithm, with each task of the latter being charged no more than $2 + L$ times.

In the second charging scheme, we use the same set of definitions. For each task in $\mathcal{T}^*(t)$, if it (the current stage) is executed by the online scheduling algorithm, then we charge the utility of the task to itself (the corresponding stage) in $\{\mathcal{T}_{alg}(t)\}_{t=1,\ldots,T}$. For each set of batched tasks in $\hat{\mathcal{T}}_1^*(t)$, we select the one in the batch with the maximum utility and form a subset. Observe that, the total utility of the subset is at most OPT. Hence, the total utility of jobs in $\hat{\mathcal{T}}_1^*(t)$ is at most $B \cdot OPT$. The same can be applied to $\hat{\mathcal{T}}_2^*(t)$. Following this scheme for all t, we charge the utilities of all the tasks executed under the offline optimal, with each task executed by the online scheduling algorithm being charged no more than $2B + 1$ times. QED.

Corollary 2.1. If each task is only one stage long, and if the online scheduling algorithm solved the local BOOSIC scheduling problem in each scheduling period optimally, then the online scheduling algorithm is 3-competitive (with respect to a clairvoyant algorithm).

Proof. This result trivially follows by substituting with $L = 1$ in the result of Theorem 2.1. QED.

2.3.4 Local Scheduling Algorithms

It remains to demonstrate how to solve the local BOOSIC scheduling problem optimally. In this section, we propose two algorithms to solve this scheduling problem. The first is

a dynamic programming-based algorithm that optimally solves it but may have a higher computational overhead. The second is a greedy algorithm that is computationally efficient but may not optimally solve the problem.

Local Dynamic Programming Scheduling Algorithm. The resource being scheduled is the GPU. Since we only consider batching together on the GPU tasks that execute the same kernel (*i.e.*, same stage on the same size input), we need to partition the scheduling interval, H, into sub-intervals where the above constraint is met. The challenge is to find an optimal partitioning. This question is broken into three steps:

- **Step 1:** Given an amount of time, $T_{j,k} \leq H$, what is the maximum utility attainable by scheduling the same stage, j, of tasks that process an input of size k? The answer here simply depends on the maximum number of tasks that we can batch during $T_{j,k}$ without violating the batching limit. If the time allows for more than one batch, dynamic programming is used to optimally size the batches. Let the maximum attainable utility thus found be denoted by $U^*_{j,k}$.

- **Step 2:** Given an amount of time, $T_k \leq H$, what is the maximum utility attainable by scheduling (any number of stages of) tasks that process an input of size k? Let us call this maximum utility U^*_k. Dynamic programming is used to find the best way to break interval T_k into non-overlapping intervals $T_{j,k}$, for which the total sum of utilities, $U^*_{j,k}$, is maximum.

- **Step 3:** Given the scheduling interval, H, what is the maximum utility attainable by scheduling tasks of different input sizes? Let us call this maximum utility U^*. Dynamic programming is used to find the best way to break interval H into non-overlapping intervals T_k, for which the total sum of utilities, U^*_k, is maximum.

The resulting utility, U^*, as well as the corresponding breakdown of the scheduling interval constitute the optimal solution. In essence, the solution breaks down the overall utility maximization problem into a utility maximization problem over time sub-intervals, where tasks process only a given input size. These sub-intervals are in turn broken into sub-intervals that process the same stage (and input size). The intuition why this division works is because the sub-intervals in question do not overlap. We pose an *order preserving* assumption on task marginal utilities with the same image size.

Assumption 2.1 (Order Preserving Assumption on Marginal Utility)**.** For two tasks τ_{i_1} and τ_{i_2} with the same size, if for one neural network stage j, we have $R_{i_1,j} - R_{i_1,j-1} \geq R_{i_2,j} - R_{i_2,j-1}$, then it also holds $R_{i_1,j+1} - R_{i_1,j} \geq R_{i_2,j+1} - R_{i_2,j}$.

Algorithm 2.1: Batching

Input: Image size index k, stage j, execution time e_b when batching b images together, batching constraint B, period H.

Output: Maximum achievable tasks $M(h)$, and optimal batch sequence $P(h)$, $\forall h \leq H$.

```
M(h) = 0, P(h) = ∅, ∀0 ≤ h ≤ H ;
for b = 1, ..., B do
    if b > M(e_b) then
        M(e_b) := b, P(e_b) := {(k, j, b)};
    end
end
for h = 2, ..., H do
    h' = arg max_{0≤h'≤h} M(h') + M(h − h') ;
    M(h) := M(h') + M(h − h') ;
    P(h) := P(h') ∪ P(h − h') ;
end
return M, P.
```

Thus, the choice of best subset of tasks to execute remains the same regardless of which stage is considered. Below, we describe the algorithm in more detail with step-by-step explanation.

Step 1: For each object size k and stage j, we can use a dynamic programming algorithm to decide the maximum number of tasks M that can execute stage j in time $T_{j,k} \leq H$. Time $T_{j,k}$ is changed between 0 and H. Observe that this computation can be done offline. The details are shown in Algorithm 2.1. (To simplify the notations, we ignore the object size and stage information here.) With the optimal number, M, computed for each, $T_{j,k}$, the corresponding utility, $U^*_{j,k}$, is simply the sum of utilities of the M highest-utility tasks that are ready to execute stage j on an input of size k.

Step 2: We solve this problem by a two-dimensional dynamic programming, where the two dimensions are the considered network stages and the time respectively. Given a time budget T_k, (for $0 \leq T_k \leq H$), Step 1 (above) already computed the optimal utility from assigning that time to only one stage. The recursive (induction) step takes as input the optimal utility from assigning some fraction of T_k to the first $j-1$ stages and the remainder to stage j, and computes the best possible sum of the two, for each T_k. Once all stages are considered, the result is the optimal utility, $U*_k$, from running tasks of input size k for a period T_k. The details are explained in Algorithm 2.2.

Step 3: Similarly to Step 2, we perform a standard dynamic programming procedure to decide the optimal time partitioning among tasks processing different input sizes. The details of this procedure, along with the integrated local dynamic programming scheduling algorithm is presented in Algorithm 2.3. With this result computed, the optimal schedule is

Algorithm 2.2: Stage Assignment.

Input: Maximum tasks M, optimal batch sequence P, available task set $\mathcal{T}_j$ for each stage j, stage count L, period H.

Output: Maximum achievable utilities U_{OPT}, and optimal batch sequence P_{OPT}, $\forall h \leq H$.

$U_{OPT}(j,h) = 0,\ P_{OPT}(j,h) = \emptyset,\ \forall j,h$;
Transitted object buffer $\mathcal{T}(j,h) = \emptyset, \forall j,h$;
for $j = 1, \ldots, L$ **do**
  for $h = 1, \ldots, H$ **do**
    if $j = 1$ **then**
      $n := \min(M(j,h), |\mathcal{T}_j|)$;
      $\mathcal{T}(j,h) :=$ n tasks with max utility in $\mathcal{T}_j$;
      $U_{OPT}(j,h) :=$ total utility of $\mathcal{T}(j,h)$;
      $P_{OPT}(j,h) := P(j,h)$;
    end
    else
      $h' := \arg\max_{h' \leq h} U_{OPT}(j-1,h') + \tilde{U}(j,h-h')$, where $\tilde{U}(j,h-h') :=$ max utility achievable with $\mathcal{T}_j \cup \mathcal{T}(j-1,h')$ in time $h-h'$;
      $\mathcal{T}(j,h) :=$ executed tasks in $\tilde{U}(j,h-h')$;
      $U_{OPT}(j,h) := U_{OPT}(j-1,h') + \tilde{U}(j,h-h')$;
      $P_{OPT}(j,h) := P_{OPT}(j-1,h') \cup P(j,h)$;
    end
  end
end
return $U_{OPT}(L,h), P_{OPT}(L,h), \forall h$.

complete.

The optimality of Algorithm 2.3 follows from the optimality of dynamic programming. Hence, the competitive ratio of the dynamic programming scheduling algorithm is 3 for single-stage task scheduling and $\min\{L+2, 2B+1\}$ for multi-stage task scheduling, according to Corollary 1 and Theorem 1, respectively. However, this algorithm may has a high computational overhead since Algorithms 2.2 and 2.3 that need to be executed each scheduling period, are $O(KLH^3)$. Next, we present a simpler local greedy algorithm, which has a better time efficiency.

Local Greedy Scheduling Algorithm. The greedy online scheduling algorithm solves the local BOOSIC scheduling problem following a simple greedy selection rule: execute the (eligible) batch with the maximum utility next. The pseudo-code of the greedy scheduling algorithm is shown in Algorithm 2.4. The greedy scheduling algorithm is simple to implement and has a very low computational overhead. We show that it achieves a comparable performance to the optimal algorithm in practice, although the two algorithms have different theoretical guarantees.

Algorithm 2.3: Local DP Scheduling Algorithm

Input: Available task set $\mathcal{T}^{(k)}(t)$ for each size, maximum tasks M, optimal batch sequence P, period H.

Output: Local task schedule x_t

for $k = 1, \ldots, K$ **do**
  $U_{OPT}^{(k)}, P_{(OPT)}^{(k)}$:=**Algorithm 2.2**$(M, P, \mathcal{T}^{(k)}(t), H)$.
end
$U_{OPT}(k,h) := U_{OPT}^{(1)}(h), \forall k, h$;
$P_{OPT}(k,h) := P_{(OPT)}(h)^{(1)}, \forall k, h$;
for $k = 2, \ldots, K$ **do**
  for $h = 1, \ldots, H$ **do**
    $h' := \arg\max_{0 \le h' \le h} U_{OPT}(k-1, h') + U_{OPT}^{(k)}(h - h')$;
    $U_{OPT}(k,h) := U_{OPT}(k-1,h') + U_{OPT}^{(k)}(h-h')$;
    $P_{OPT}(k,h) := P_{OPT}(k-1,h') \cup P_{OPT}^{(k)}(h-h')$;
  end
end
return *The schedule* x_t *according to* $P_{OPT}(K,T)$.

Algorithm 2.4: Local Greedy Scheduling Algorithm

Input: Available task set $\mathcal{T}(t)$, the limitation of non-overloading batch size $B^{(k)}$ for each image index k.

Output: Local task schedule x_t

while *until the end of the period* **do**
  for $k = 1, \ldots, K$ **do**
    $\mathcal{T}_k(t)$:= set of available tasks of size k.
    if $|\mathcal{T}_k(t)| \le B^{(k)}$ **then**
      $U_k(t)$:= total utility of tasks in $\mathcal{T}_k(t)$.
      $\tilde{\mathcal{T}}_k(t) := \mathcal{T}_k(t)$
    end
    else
      $\tilde{\mathcal{T}}_k(t)$:=$B^{(k)}$ tasks with the maximum utility in $\mathcal{T}_k(t)$, $U_k(t)$:= total utility of tasks in $\hat{\mathcal{T}}_k(t)$.
    end
  end
  Execute the tasks in $\tilde{\mathcal{T}}_k(t)$ with the maximum value of $U_k(t)$.
end
return x_t

2.4 IMPLEMENTATION

In this section, we briefly introduce the implementation details of the proposed scheduling framework. An overview of the proposed scheduling framework implementation is demon-

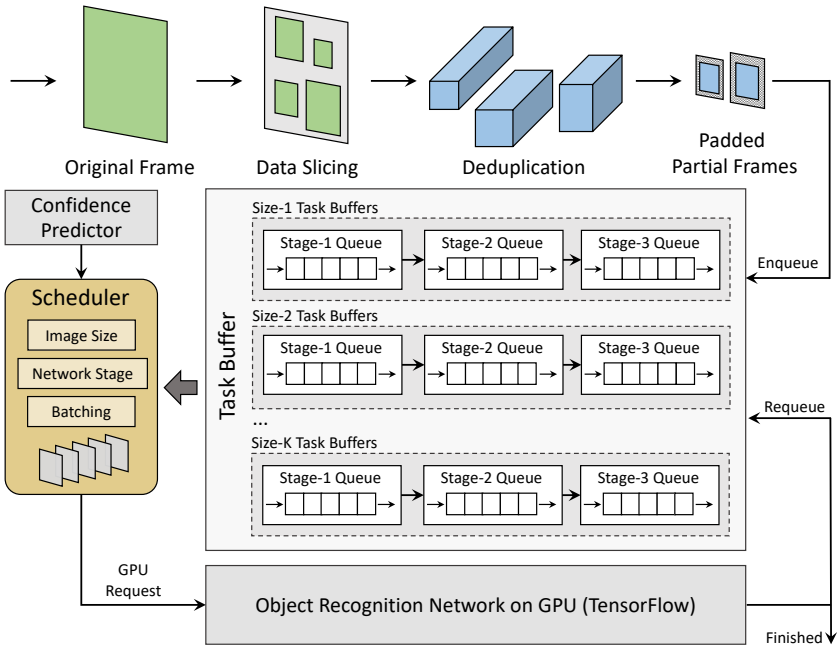


Figure 2.4: System architecture for the proposed scheduling framework. All components are implemented in the user space, and the scheduled batch of images is submitted as a single GPU request.

strated in Figure 2.4.

The scheduled task is defined as the recognition of individual objects by a state-of-the-art convolutional neural network (CNN), namely the residual neural network (ResNet), which is implemented in TensorFlow [29]. To store the arrived but not finished tasks, we define a feature buffer for each (*image size, network stage*) pair. For a given image size, the buffer for each stage is intrinsically a priority queue to store the tasks waiting to execute this stage. The priority of each task is defined as its predicted marginal utility for the stage to execute. When two tasks have the same marginal utility, the one with an earlier deadline will be prioritized. When a new frame arrives, it first goes through a data slicing step, assisted by the LIDAR input, to extract the partial frames. The useless background area is removed. After filtered by the deduplication module, partial frames are padded to their closest target sizes with black borders. Finally, we push tasks into the stage queues for their corresponding buffers. Similarly, when the tasks finish one stage of execution, they will be pushed into the next stage queue unless they are finished or overdue. Besides, we periodically clean up outdated tasks from each buffer to save the memory space.

Scheduling within NVIDIA GPU drivers are quite restrictive, so we follow the idea by Yao *et al.* in [27] to implement the scheduler as a middleware service in user space. It first collects the status from task buffer for each (image size, network stage) pair, which is then

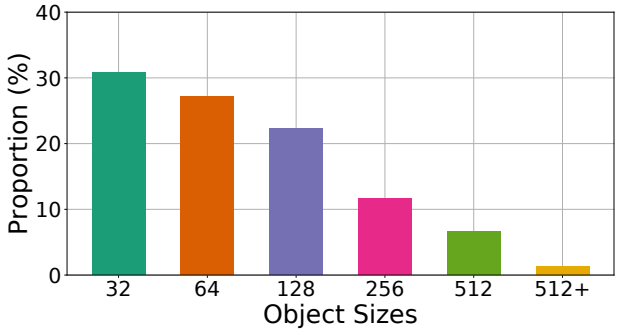


Figure 2.5: Waymo object bounding box size distribution.

used as the input to our scheduling algorithm. The scheduling output tells us which image size and network stage to execute next, as well as the number of tasks to batch. Then it collects feature maps from the corresponding task buffer, and submit the batch as a single GPU request (kernel). This operation would lead to an extra memory swap between CPU and GPU on desktop machines. However, the integrated GPU of NVIDIA Jetson Xavier SoC shares the same memory with the CPU, so that the extra time delay is acceptable here.

2.5 EVALUATION

In this section, we verify the effectiveness and efficiency of our proposed scheduling framework by comparing it with several state-of-the-art baselines on a large-scale self-driving dataset, Waymo Open Dataset.

2.5.1 Experimental Setup

Hardware Platform: All experiments are conducted on an NVIDIA Jetson AGX Xavier SoC, which is specifically designed for automotive platforms. It's equippet!d with an 8-core Carmel Arm v8.2 64-bit CPU, a 512-core Volta GPU, and 32 GB memory. Xavier delivers over 30 TOPS for deep learning applications while consuming less than 30 Watts [30]. Its mode is set as MAXN with maximum CPU/GPU/memory frequency budget, and all CPU cores are online.

Dataset: Our experiment is performed on the Waymo Open Dataset [31], which is a large-scale autonomous driving dataset collected by Waymo self-driving cars in diverse geographies and conditions. It includes driving video segments of 20s each, collected by LiDARs and cameras at 10 Hz. All LiDAR and camera data are synchronized. The object classes are limited to 4 classes: vehicle, pedestrian, cyclist, and sign. Only the front camera data is used in our experiment. We show the distribution of object (bounding box) sizes in Figure 2.5.

The figure depicts the length of the longer side, rounded up to the preset bins: 32, 64, 128, 256, and 512. This also reflects the practical object size distribution from the driver's vision. Since we do not need the added resolution for identification, in our experiment, objects with size larger than 256 are down-scaled to 256 while preserving its aspect ratio. All remaining images are padded to the target size bins.

Neural Network Training: We use ResNet proposed by He *et al.* [19] for object classification. The network is trained on a general-purpose object detection dataset, COCO [32]. It contains 80 object classes that include those of the Waymo dataset.

Scheduling Load and Evaluation Metrics: We extract the distance between objects and the autonomous vehicle (AV) from the projected LiDAR point cloud. The deadlines of object classification tasks are set as the time to collision (TTC) with the AV. To simulate different loads for the scheduling algorithms, we manually change the sampling period (*i.e.*, frame rate) from 40ms to 160ms. Since actual frame capture was done at 100ms intervals, the above corresponds to replaying the world in "slow motion" to "fast-forward" mode to understand the impact of speed on the ability of the perception subsystem to keep up. We consider a task to miss its deadline if the scheduler fails to run the mandatory part of the task by the deadline. Otherwise, we consider the task to return a timely but possibly imprecise result. In the following evaluation, we present both the *normalized accuracy* and *deadline miss rate* for different algorithms. The normalized accuracy is defined as the ratio between achieved accuracy and the maximum accuracy when all neural network stages are finished for every object.

2.5.2 Compared Scheduling Algorithms

The following scheduling algorithms are compared.

- **OnlineDP**: the online scheduling algorithm we proposed in Section 4.3. The local scheduling in each period is conducted by the hierarchical dynamic programming algorithm.
- **Greedy**: the online scheduling algorithm we proposed, with the local scheduling conducted by greedy batching algorithm.
- **Greedy-NoBatch**: It always execute the object with maximal marginal utility. No batching is performed for this algorithm.
- **EDF:** It always chooses the task stage with the earliest deadline (without considering task utility).

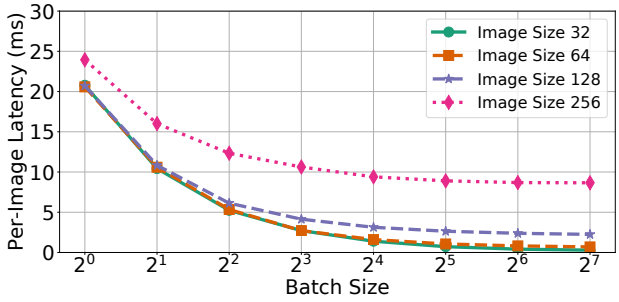


Figure 2.6: Per-image latency of ResNet on NVIDIA Jetson Xavier SoC, with respect to different image sizes and batch sizes.

- **Non-Preemptive EDF (NP-EDF):** Unlike regular EDF, this algorithm does not allow preemption. Once a task starts executing, it continues until it is finished or its deadline is reached. It is included to understand the impact of allowing preemption on stage boundaries compared to not allowing it.
- **FIFO:** It runs the task with the earliest arrival time first. All stages are performed as long as the deadline is not violated.
- **RR:** Round-robin scheduling algorithm. Runs one stage of each task in a round-robin fashion.

2.5.3 Neural Network Time Profiling

We first profile the inference time of ResNet on the NVIDIA Jetson Xavier SoC, for varying image sizes and batch sizes. The result is shown in Figure 2.6. We can observe that when the image size is small (*e.g.*, 32×32 or 64×64), increasing the batch size is always beneficial and leads to a lower per-image latency. When the image size becomes larger (128×128 or 256×256), the benefit of parallelism gradually decreases. This is because the GPU is fully utilized at maximum parallelism, so increasing batch size increases execution time proportionally. Accordingly, we set the batching limit for each image size to be the batch size beyond which the per-image inference time stops decreasing. This size is 128, 128, 32, 8 for the four image sizes 32, 64, 128, and 256, respectively. The execution time for each valid batch is below 100ms.

2.5.4 Slicing and Batching

Next, we compare the inference time for *full frames* and *batched partial frames with/out deduplication*. In full frame processing, we directly run the neural network on image-captured

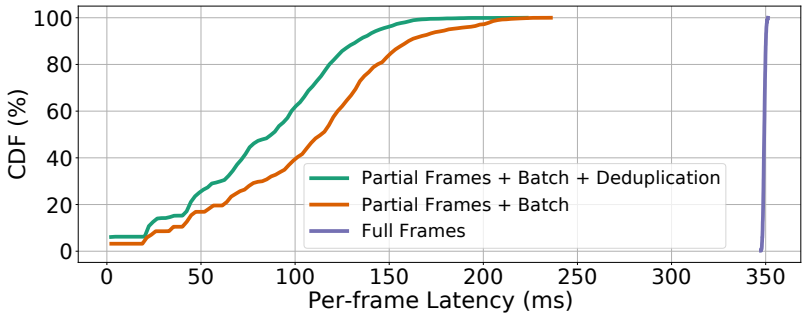


Figure 2.7: Cumulative distribution of end-to-end latency on full frames with three methods. The execution time for frame slicing, deduplication (if applicable), batching, and neural network inference are all counted.

full images, whose size is 1920 × 1280. In *batched partial frames*, we do the slicing into bounding boxes within one frame first, then perform the deduplication (if applicable), and finally batch execution of objects with same size. Each frame is evaluated independently. No imprecise computation is considered. The end-to-end latency for each full frame, including both preprocessing and network inference time, is reported here. Our results show that the average latency for full frames is 350 ms, while the average latency for (the sum of) batched partial frames is 105 ms without deduplication, and 83 ms with deduplication. Besides, the cumulative distributions of frame latencies for the three methods are shown in Figure 2.7. We can see that by batched partial frames (no deduplication), most cases have a latency below 200 ms, while deduplication further decreases the latency for most cases below 150 ms. Data slicing, batching, and deduplication steps, although induce extra processing delays, can effectively reduce the end-to-end latency. However, neither approach is fast enough compared to 100 ms sampling period, so that the imprecise computation model and prioritization are needed.

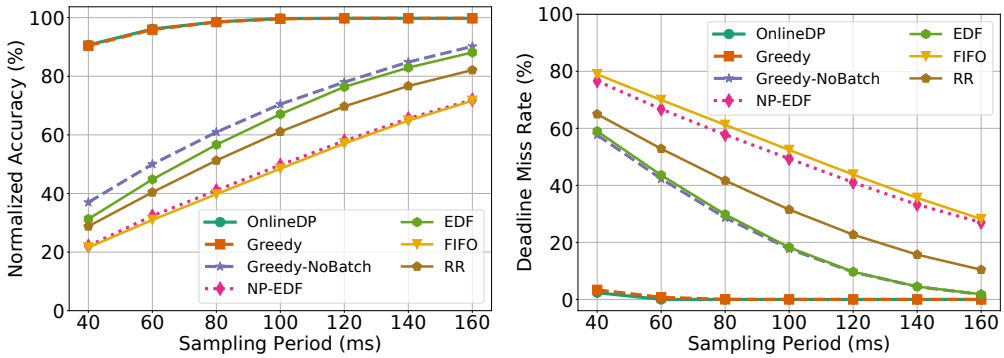
2.5.5 Scheduling Policy Comparisons

Next, we evaluate the scheduling algorithms in terms of achieved classification accuracy and deadline miss rate. We change the replayed camera frame rate to vary the load. To isolate the effect of real-time prioritization from that of object deduplication, we turn off the latter in this part. The scheduling results are presented in Figure 2.8. The two proposed algorithms, OnlineDP and Greedy, clearly outperform all the baselines with a large margin in all metrics. The improvement comes for two reasons: First, the integration of the imprecise computation model into neural networks makes the scheduler more flexible. It makes the neural network partially preemptive at the stage level, and gives the scheduler an extra

degree of freedom (namely, deciding how much of each task to execute). Among stage-level scheduling algorithms, Greedy-NoBatch shows a similar deadline miss rate to EDF, but has a better accuracy. Since Greedy-NoBatch is unable to predict confidence in (*i.e.*, utility of) future stages until it has executed one, it has no inherent way of deciding which task to start first. Thus, we select the task with the earliest deadline to execute first without violating the maximum utility rule. Second, the involvement of batching mechanism simultaneously improves the model performance and alleviate deadline misses. The batching mechanism enables the GPU to be utilized at its highest parallel capability. The deadline miss rates of both OnlineDP and Greedy are pretty close to 0 under any task load. We can also see that Greedy shows similar performance as OnlineDP, though they possess different theoretical results. One practical reason is that the utility prediction function can not perfectly predict the utility for all future stages, where the OnlineDP scheduling can be negatively impacted. Instead, Greedy only relies on the utility prediction for the next stage to make the scheduling decision.

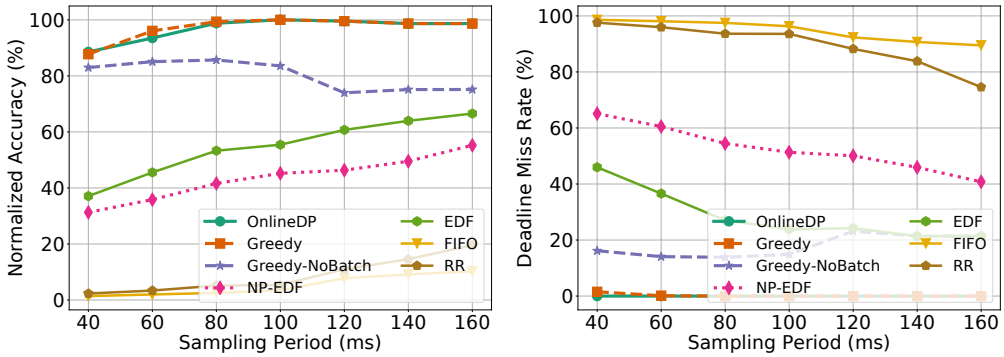
NP-EDF and FIFO have similar performance in this experiment, which might seem like a "bug". Upon closer inspection, we realize that this is because the objects in the vehicle's field of view have similar deadlines most of the time, making FIFO similar to EDF. This is expected because most of the time, when driving, no abnormal events occur that require an abrupt preemption of attention to a more critical object. While less common, such instances, however, are very important. Response to such instances is precisely what distinguishes good driving from bad driving.

To evaluate scheduling performance in driving scenarios involving the aforementioned important subcases, we compare the metrics of different algorithms for the subset of "critical objects". Critical objects are defined as objects whose time-to-collision (and hence processing deadline) fall within 1s from when they first appear in the scene. Results are shown in Figure 2.9. We notice that the accuracy and deadline miss rate of FIFO and RR are much worse in this case (because severe priority inversion occurs in these two algorithms). The deadline-driven algorithms (NP-EDF and EDF) can effectively resolve this issue because objects with earlier deadlines are always executed first. However, their general performance is limited for lack of utility optimization. The utility-based scheduling algorithms (Greedy, Greedy-NoBatch, and OnlineDP) are also effective in removing priority inversion, while at the same time achieving better confidence in results. These algorithms multiply a weight factor $\alpha > 1$ to increase the utility of handling critical objects, so that they are preferred by the algorithm over non-critical ones. We empirically found that $\alpha = 10$ gives good results. A more detailed exploration of this hyper-parameter will be presented in an extended report (but removed here for space limitations).



(a) Normalized accuracy. (b) Deadline miss rate.

Figure 2.8: Accuracy and deadline miss rate comparisons on all objects.

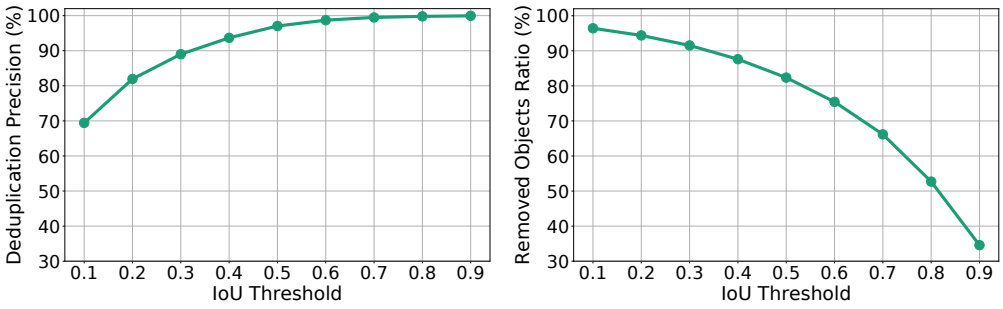


(a) Normalized accuracy of critical objects. (b) Deadline miss rate of critical objects.

Figure 2.9: Accuracy and deadline miss rate comparisons on critical objects. Critical objects are defined as objects that have a deadline less than 1s.

2.5.6 Impact of Deduplication

In this part, we report results of the deduplication module. The overlap between bounding boxes from consecutive frames is evaluated by computing their area intersection over union (IoU) score, which is defined as the ratio between their intersection area and union area. We seek to explore the best threshold for deduplication (*i.e.*, for considering two bounding boxes to be referring to the same object). To do so, we change the IoU threshold from 0.1 to 0.9 and compute deduplication precision, define as the percentage of time the box considered to be a duplicate was indeed an image of the same object. The results are shown in Figure 2.10. We can precisely identify 99.5% bounding boxes belonging to the same object using a threshold of 0.7; while 99.95% of removed bounding boxes refer to the same object when the threshold is 0.9. We also report the ratio of saved workload (*i.e.*, removed objects) in Figure 2.10(b). When the threshold is 0.9, we can reduce the total workload by 34.6%, while using a threshold of 0.7 can lead to 66.7% bounding boxes being removed.



(a) Deduplication precision. (b) Removed bounding box ratio.

Figure 2.10: Scheduling results of Greedy algorithm after applying deduplication.

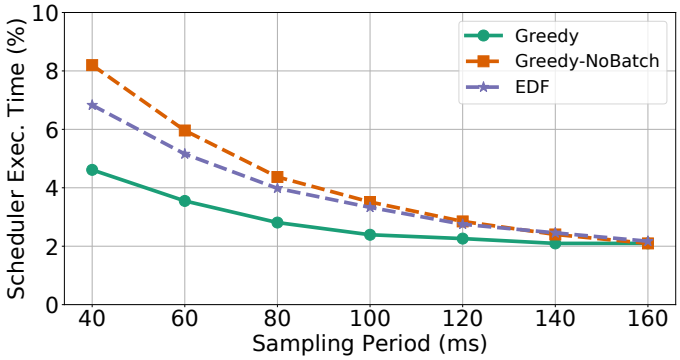


Figure 2.11: Scheduling algorithm execution time comparisons.

2.5.7 Scheduling Algorithm Execution Time

In this part we evaluate the execution time of our proposed scheduling algorithm. Note that while the scheduled tasks run on the GPU, the scheduler runs on a single CPU core. As we mentioned in a previous statement, the OnlineDP scheduling algorithm is too slow to be applied in real time scenarios, and no computation is needed for FIFO and RR in addition to a queue. Thus, we only compare the Greedy, Greedy-NoBatch, and EDF. Specifically, we record the execution time of the scheduler on the CPU core as a percentage of the execution time of the neural networks on the GPU. Figure 2.11 shows the results. We can see that when the sampling period is large (*i.e.*, inverse of frame rate), all three algorithms show similar latency (around 2%), likely attributed to other CPU overheads. When the task load becomes larger, Greedy shows a better time efficiency than Greedy-NoBatch and EDF. The reason is that the batching mechanism effectively consumes more objects in the given time, so that the size of object buffer in Greedy is much smaller than Greedy-NoBatch and EDF. The execution time of our Greedy algorithm is within 5% in all evaluated workloads. Since the scheduling algorithm is executed on single CPU core, we can find that CPUs are idle most of the time. The bottleneck is indeed the GPU.

2.6 INVESTIGATING THE IMPACT OF CRITICALITY DESIGN

The scheduling objective defined in the vanilla BOOSIC problem aims to optimize the overall confidence in model predictions. It allocates computational resources to tasks whose execution attains the largest increase in confidence. Tasks are split into two groups (critical or non-critical) by thresholding on object distance. Critical tasks are assigned a relatively high but fixed weight.

In this section, we generalize it by considering a fine-grained task-specific criticality weight w_i for each task that is multiplied by confidence. If we set the weight factor $w_i = 1$ for each task, then the algorithm always prioritizes tasks with high confidence increase. On the contrary, if we set the predictive confidence $x_i = 1$ for each task, then the task execution order strictly complies with task criticality. By considering both factors, we achieve more effective utilization on limited computation resources: We allocate resources preferentially to more critical tasks that also observe a substantial increase in prediction confidence.

We instantiate two different mechanisms for computing weight, w_i, in this paper: distance-based criticality, and relative velocity-based criticality. They are discussed below.

2.6.1 Distance-based Criticality

As a straightforward instantiation, we first propose and use distance-based criticality. It is tempting to assume that (in a purely distance-based criticality assignment algorithm) closer objects should monotonically receive a higher priority because they induce a higher risk of future collisions. As suggested by one of our anonymous reviewers, however, this reasoning is flawed. Objects closer than a certain minimum threshold, l_{min}, might already be too close to comfortably avoid. It is thus imperative to account for them while they are are still sufficiently far way. In other words, the value function for determining criticality is *shifted*. Namely, assume the distance of the i-th object is l_i and the maximum LiDAR detection range is l_{max}, the distance-based criticality weight is defined as:

$$w_i = \begin{cases} 0, & \text{if } l_i \leq l_{min}; \\ \frac{1}{\left(\frac{l_i - l_{min}}{l_{max} - l_{min}}\right)^k + \epsilon}, \ k \geq 1, & \text{if } l_i > l_{min}, \end{cases} \tag{2.6}$$

where ϵ is a small positive term for stabilization. The threshold l_{min} is the minimal safe object distance, defined as:

$$l_{min} = v \cdot H\delta + \frac{v^2}{2 \cdot a}, \tag{2.7}$$

where v is the current velocity of the AV, and a is the largest acceleration of the AV when it makes a hard brake. Equation (2.7) computes the minimum safe distance l_{min}, in a velocity-dependent fashion, for the vehicle to avoid an obstacle. Below that distance, a safety-critical override (*e.g.*, a collision avoidance system) should immediately intervene and stop the car or reduce speed (to increase l_{min}). Note that, if the car is stopped, l_{min} is zero. We henceforth call l_{min} a (distance) *shift point*, since the weight assignment, w_i is effectively shifted by l_{min} to focus on more distant objects.

One limitation of the above design is that it does not account for the velocity of the other objects. In general, an object that is decelerating might be more of a concern than one that is accelerating (away from the AV), even if both are presently the same distance away. Below, we describe a modification of the above algorithm that accounts for relative velocity.

2.6.2 Relative Velocity-based Criticality

We can estimate relative velocity information by computing distance change between consecutive frames. We need an efficient object tracking module to calculate relative velocity of surrounding objects. Its intuition is same as the time-to-collision metric based on the relative velocity in navigation services. We follow the SORT algorithm proposed by Bewley *et al.* [33] for object tracking. It adopts a tracking-by-detection strategy to track multiple objects in parallel. At each frame, we try to map each bounding box to an existing object track through a data association algorithm (the Hungarian algorithm [34] in our implementation), based on the intersection-over-union (IoU) matrix between new bounding boxes and predicted locations for object tracks. The assumption is that, if a new bounding box is highly overlapped with the projected location of a previous object according to the motion model extracted from its past trajectory, then we believe they belong to the same object. No semantic information (*i.e.*, category of the object) is needed during the mapping. The relative velocity is:

$$\tilde{v}_i = \frac{l'_i - l_i}{H\delta}, \tag{2.8}$$

where l_i and l'_i denote the distance of object i in current period and the previous period respectively. A positive value means the object is approaching while a negative value means the object is moving away. We further remove object mappings that lead to an abnormally high relative velocity. We correspondingly define the object deadline as:

$$d_i = \begin{cases} d_{max} & \text{if } \tilde{v}_i \leq 0 \text{ or } \frac{l_i}{\tilde{v}_i} > d_{max}; \\ \frac{l_i}{\tilde{v}_i} & \text{if } 0 < \frac{l_i}{\tilde{v}_i} \leq d_{max}, \end{cases} \tag{2.9}$$

where $d_{max} = \frac{l_{max}}{v_o}$ is defined as the maximum deadline. It is calculated using the maximum LiDAR range l_{max} and the observer velocity v_o. The object weight is:

$$w_i = \begin{cases} 0 & \text{if } d_i \leq d_{min}; \\ \frac{1}{\left(\frac{d_i - d_{min}}{d_{max} - d_{min}}\right)^k + \epsilon}, \ k \geq 1, & \text{if } d_i > d_{min}. \end{cases} \tag{2.10}$$

As in the previous section, we assume when an object deadline falls below the threshold d_{min}, the safety-critical system should immediately interfere and take corresponding action. We call d_{min} a (deadline) shift point.

One limitation of this mechanism is that we need at least two appearances of the same object to calculate its relative velocity. In the data association step, if the new bounding box can not be mapped to any existing tracks, then it is considered as a potential new object. In this case, we assume the object to be static.

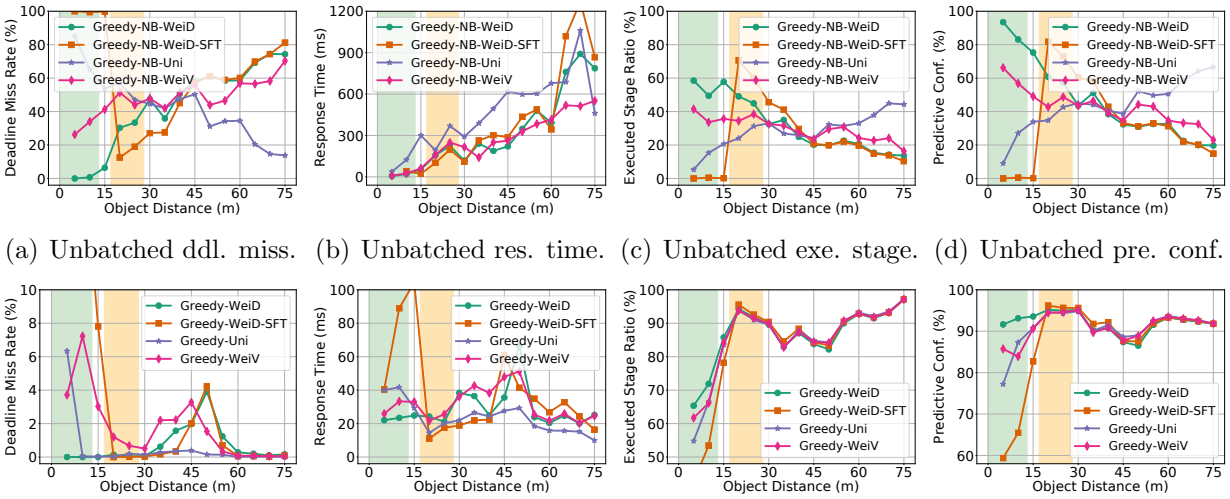
2.6.3 Evaluation on Criticality Design

In this subsection, we evaluate the impact of the proposed criticality designs on the achieved prioritization effect. The following scheduling algorithms are evaluated:

- **Greedy-Uni**: This is a simplification of the online scheduling algorithm, wherein the local scheduling conducted by an unweighted but batched greedy algorithm. The task weight is uniformly set as $w_i = 1$ for all tasks. Its scheduling objective is also equivalent to (approximately) maximizing the achieved model accuracy on all objects, without differentiating the object criticality.

- **Greedy-WeiD/WeiV**: This is the proposed online scheduling algorithm above, with the local scheduling conducted by the weighted and batched greedy algorithm. Its scheduling objective is a tradeoff between prioritizing execution for critical objects, and the overall model accuracy on all objects. In the following experiments, we use -WeiD to denote the distance based-criticality assignment and use -WeiV to denote the relative velocity-based criticality assignment. The default implementation neglects the shift point (*i.e.*, sets l_{min} and d_{min} to zero). We use -SFT to denote algorithm variations with a non-zero shift point (*i.e.*, $l_{min} \neq 0$ or $d_{min} \neq 0$).

- **Greedy-NB**: It always executes the single (task, stage) with maximal marginal utility. No batching is performed. Utility is set proportional to the achieved predictive confidence. In other words, task weight is uniformly set to $w_i = 1$, for all tasks.



(a) Unbatched ddl. miss. (b) Unbatched res. time. (c) Unbatched exe. stage. (d) Unbatched pre. conf.

(e) Batched ddl. miss. (f) Batched res. time. (g) Batched exe. stage. (h) Batched pre. conf.

Figure 2.12: Scheduling results of greedy algorithms on objects with **different distances**, when applying distance-based weight (**WeiD**), relative velocity-based weight (**WeiV**) and uniform (**Uni**) weight. We separately report the results with and without task batching. The range highlighted with green shows the advantage of **Greedy(-NB)-WeiD**, and the range highlighted with orange shows the advantage of **Greedy(-NB)-WeiD-SFT**.

- **Greedy-NB-WeiD/WeiV**: Same as **Greedy-NB**, except that the weight, w_i is calculated according to distance-based criticality (-WeiD) or velocity-based criticality (-WeiV).

We implement and evaluate the distance-based criticality assignment and relative velocity-based criticality assignment, and separately compare their different versions, including a weight shift corresponding to the minimum distance l_{min} and the minimum relative deadline $\frac{d_{min}}{d_{max}}$ respectively. In following experiments, we simulate the critical driving scenarios by setting the frame replay period to 60ms. Four metrics are compared: 1) Deadline miss rate; 2) Mean response time (*i.e.*, the execution time of the first network stage); 3) Mean executed stage (*i.e.*, the average ratio of executed network stage over its maximum network stage); 4) Mean predictive confidence (*i.e.*, the mean ratio of achieved predictive confidence over the maximum confidence when all stages are executed). In the last two metrics, we compute the relative ratio to resolve the impact of specific task instances.

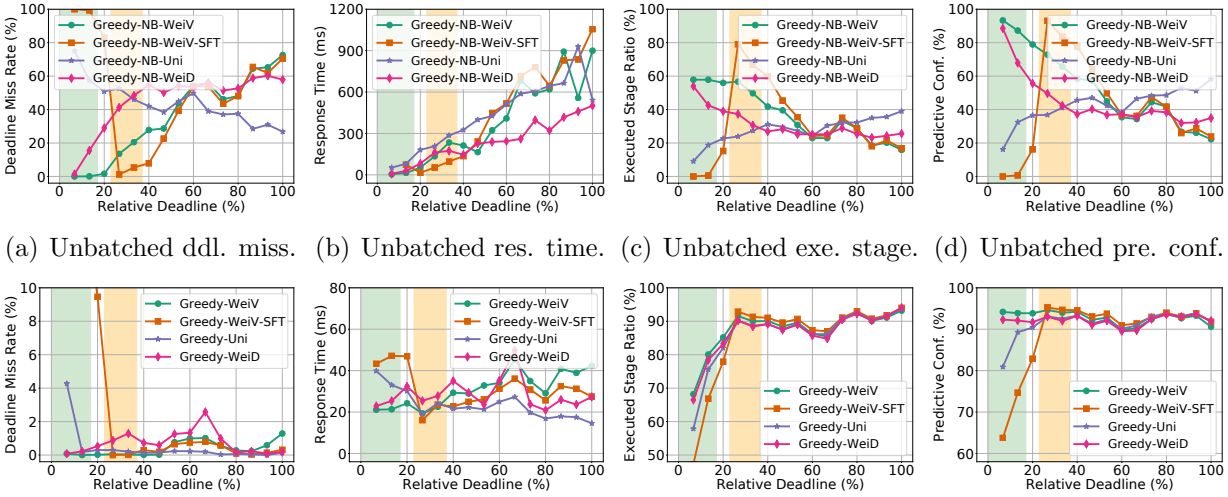
Case 1: Distance-based Criticality. We compare the scheduling results on objects at every distance range when different weight mechanisms are applied in Figure 2.12. We first look at the greedy algorithms without batching. When using uniform utility, objects are not differentiated according to their distances besides the predictive confidence, so we see close objects show much higher deadline miss rate and lower predictive confidence. Insufficient

computation resources are assigned to close objects within their short deadline, who are actually more critical to the system safety. The issue is resolved in weighted greedy by partially trading the performance of distant objects. Close objects are strictly prioritized over distant objects since no task batching is applied. For close objects, significantly more computation resources are allocated, leading to the significant decrease in deadline miss, and increase in prediction quality (reflected in both executed stage and predictive confidence). The shifted version also properly adjusts the focus (*i.e.*, prioritization) of the algorithm as expected. The prioritized distance range moves from 0-10m (highlighted in green) to 15-25m (highlighted in orange) after applying the shift. We can conclude that the weighted greedy algorithm without batching perfectly solve the problem of priority inversion, but its general performance is inferior.

Then we also compare the batched algorithms. Since batching significantly increases the task processing capacity of GPU, the deadline misses and response times of all objects are significantly reduced in the batched greedy algorithm with uniform-weight. However, there are still deadline misses on very close objects (5m) in Greedy-Uni. Our objective is to resolve deadline misses at close objects, and increase their prediction quality, considering their importance to the system safety. Misclassification at such objects can lead to serious safety issues. After applying the weight mechanism, the deadline misses on close objects are resolved, and their response times are also further reduced. More executed stages and higher predictive confidence on close objects indicate that more computation resources are allocated to them by Greedy-Wei compared to Greedy-Uni. The improvement comes at the cost of increased deadline miss and degraded prediction quality at distant objects (*i.e.*, 45-60m). In addition, the shifted algorithm successfully ignore the closest objects (*i.e.*, $\leq$ 15m) and prioritizes objects between 15-30m. Note that we set l_{min}=15m for better visualization effect.

Finally, we briefly analyze the performance of Greedy-WeiV w.r.t objects at different distance ranges. Greedy-WeiV, though being better than Greedy-Uni in prioritizing close objects because objects with short relative deadlines are mostly close, is not optimal as Greedy-WeiD because it can prioritize far objects as well if they have a fast relative velocity. In conclusion, Greedy-WeiD is the best option if we simply want to prioritize the closest objects.

Case 2: Relative Velocity-based Criticality. Next we compare the scheduling performance when relative velocity-based criticality is applied. The associated results are presented in Figure 2.13. In this experiment, we use the relative deadline $\frac{d_i}{d_{max}}$ as the x-axis, which is decided by both the object distance and its relative velocity. In the shifted weighted greedy algorithm, we set the shift $\frac{d_{min}}{d_{max}} = 20\%$. Without applying the weight mechanism,



(a) Unbatched ddl. miss. (b) Unbatched res. time. (c) Unbatched exe. stage. (d) Unbatched pre. conf.

(e) Batched ddl. miss. (f) Batched res. time. (g) Batched exe. stage. (h) Batched pre. conf.

Figure 2.13: Scheduling results of greedy algorithms on objects with **different relative deadlines**, when applying relative velocity-based weight (**WeiV**), distance-based weight (**WeiD**), and uniform (**Uni**) weight. We separately report the results with and without task batching. The range highlighted with green shows the advantage of **Greedy-WeiV**, and the range highlighted with orange shows the advantage of **Greedy-WeiV-SFT**.

Greedy-Uni still has some deadline misses on critical objects (*i.e.*, relative deadline $\leq 20\%$, as highlighted in green), even though the values are already much lower than Greedy-NB-Uni. The application of relative velocity-based criticality effectively reduces the deadline miss rate of critical objects to zero. Both response efficiency and prediction quality (*i.e.*, executed stages and prediction confidence) on critical objects with short relative deadline are improved after applying the relative velocity-based weight mechanism. In addition, Greedy-WeiV-SFT successfully skips the objects with deadlines below d_{min}, and only prioritizes those with their deadlines larger than but close to d_{min}, as highlighted in orange. Regarding the cost, Greedy-WeiV induces slightly higher deadline miss rate and longer response times on objects with long deadlines, compared to the Greedy-Uni. The prediction qualities are quite similar between Greedy-WeiV and Greedy-Uni on objects with long relative deadlines.

We further investigate the performance of Greedy-WeiD in this experiment. It presents small deadline miss rates between 5% to 40%, where Greedy-WeiV shows no deadline miss. The average response time and prediction quality of Greedy-WeiD is also inferior to Greedy-WeiV in this range. Therefore, we can conclude that although Greedy-WeiD shows similar performance as Greedy-WeiV in some cases (*e.g.*, both give no deadline miss on objects with the shortest relative deadlines), it can not replace Greedy-WeiV to provide timely and high-quality responses to all fast-approaching objects.

2.7 INVESTIGATING THE IMPACT OF IMAGE RESIZING

In this section, we extends the previous scheduling framework by adopting an image resizing mechanism as a substitution for previously used imprecise computation model. This extension is motivated by the observation by Torralba [35] that there is a threshold on the resolution of images deciding whether they are recognizable or not by human eyes. We believe a similar phenomenon also exists for machine perception models, where the extra resolution far above the threshold might not be beneficial. Instead, we can safely downsize these images such that the processing efficiency can be substantially improved, without any degradation on the model accuracy.

2.7.1 Imprecise Computation vs. Model Switching

In order to understand the inherent problem with the imprecise computation model for neural classification, we observe (as discussed in [36]) that each layer of a deep neural network extracts an abstraction of the input features from the previous layer. Generally, the deeper the network goes, the higher the level of abstraction will be. When multiple outputs can be extracted from different depths of one neural network, it is impossible to train the neural network in a manner that is simultaneously optimal for every choice of depth. Instead, the training loss is usually defined as a weighted sum of loss over all allowed choices of network depth. Thus, in essence, each layer is optimized for a compromise among multiple settings of possible model depth. As a result, the training will not be optimal for any specific choice of depth, and the prediction accuracy will be accordingly affected. A neural network trained specifically for a given depth will perform better for that depth. Thus, for example, a smaller neural network trained specifically for a given depth will achieve a higher accuracy than a partially executed larger neural network, given the same number of executed stages.

In short, we argue for *switching among different neural network models*, each individually trained for a different point in the latency/quality design trade-off, as opposed to training a single model that can be adapted at runtime. The input resizing approach adopted in this paper is consistent with the aforementioned guiding principle. It allows the scheduler to select an input size (and thus a corresponding neural network size) in a criticality-aware manner that attains the desired trade-off. We validate this reasoning in the evaluation. The disadvantage, of course, is that (when using model switching) one needs to commit to the network used for each input at the time that the processing of that input starts. One cannot switch models once some stages have already been executed, as it will lead to loss of partial results computed by the switched-out model thus far. In contrast, in the imprecise

computation model, the ultimate number of layers executed can be adapted for a given input even after the processing of that input starts. We show that this latter flexibility is overshadowed by the overall quality reduction that arises from inability to optimize neural network parameters for a given network depth.

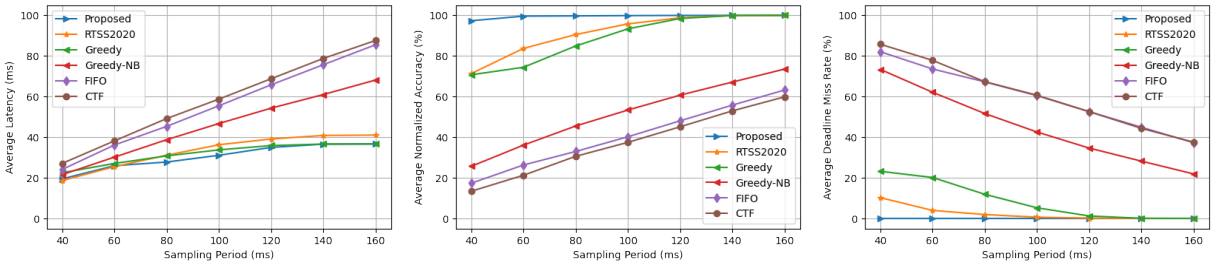
From the model switching perspective, we formally formulate the scheduling problem as an optimal resizing problem, and correspondingly propose an approximated batched greedy solution to schedule the task processing. The details of the problem formulation and the algorithm are skipped here, but can be found in [37].

2.7.2 Evaluations between Image Resizing and Imprecise Computation

We compare the performance of different scheduling algorithms under the same settings, with different frame intervals. The compared algorithms include:

- **Proposed**: The proposed greedy scheduling algorithm with resizing. It uses the greedy heuristic to choose a near-optimal subset of tasks and their corresponding new input sizes, and batches the tasks with the same new size together for execution.
- **RTSS2020**: The greedy scheduling algorithm with staged neural networks proposed by Liu *et al.* [38]. It uses an imprecise computation model with a mandatory part and several optional parts.
- **Greedy**: The proposed scheduling algorithm with resizing turned off. It chooses the near-optimal subset of tasks at their original input sizes and batches the tasks with the same size together for execution.
- **Greedy-NB**: It chooses the tasks with the highest utility/weight value first, and always processes inputs at their original sizes. Inputs of the same size are not batched together.
- **FIFO**: It executes the tasks that arrive earlier first, and always process tasks with their original sizes. The tasks of the same size are not batched together.
- **CTF**: It always chooses the most critical task first, and always processes tasks with their original sizes. The tasks of the same size are not batched together.

We evaluate the scheduling algorithms in terms of achieved classification accuracy, average latency, and deadline miss rate. A deadline is considered missed if a task does not execute (*i.e.*, $s = NULL$), *and* the object has not been seen before. This is as opposed to a situation



(a) Average inference latency. (b) Average normed accuracy. (c) Average deadline miss rate.

Figure 2.14: Evaluation results comparison on *image resizing* vs. *imprecise computation*.

where a task is not executed because tracking decides to inherit object classification from the previous frame. This way, we do not penalize the algorithm for skipping a task due to temporal redundancy. The results are presented in Figure 2.14. The Proposed algorithm outperforms the other algorithms by a clear margin in terms of both deadline misses and normalized accuracy, especially when the frame interval is shorter. It achieves no deadline misses for all tested frame intervals, and maintains high accuracy and low latency. The improvement comes for several reasons: First, the ability to choose from different image sizes enables the scheduler to trade off inference quality and execution time; Second, the ability to batch a larger number of tasks together improves the utilization of the GPU; Third, the smaller models have a higher efficiency than partial executions of the larger models.

Note that the difference between Proposed and Greedy is attributed solely to image resizing. As can be seen, resizing improves accuracy and reduces deadline misses at the same time. Similarly, the difference between Proposed and RTSS2020 is attributed to the relative advantage of image resizing compared to imprecise computations. In contrast, the difference between Greedy and Greedy-NB is attributed to batching. As one might imagine, removing batching substantially drops performance. Finally, the difference between Greedy-NB and CTF/FIFO is attributed to considering latency in the optimization.

The average batch size for Proposed and RTSS2020 is shown in Figure 2.15. The lowest tested frame interval is set to 10ms. The average batch size for Proposed is always larger than RTSS2020, and can reach up to more than 18 when the frame interval is 10 ms. While the average batch size for RTSS2020 also increases as the frame interval shortens, it is not because the algorithm actively does so, as RTSS2020 can only batch the tasks with the same original sizes together. Like all the other algorithms, the obtained optimal execution sequence for RTSS2020 does not change with the frame interval. The optional neural network stages are not executed because of hitting the time limit. The increase in batch size has the same reason as the increase in deadline miss rate: the batches with smaller utility, usually

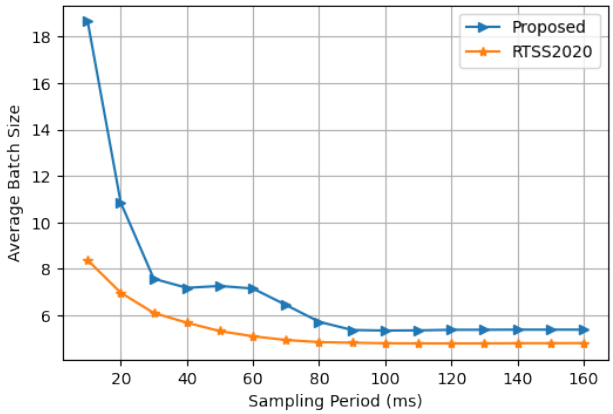


Figure 2.15: Average batch size comparison.

those with a smaller batch size, are not prioritized and cannot be executed within the frame interval. The Proposed algorithm on the other hand, makes different scheduling decisions for different frame intervals. For the most extreme case, it can resize all the tasks to the most efficient 32×32 size and process them in one batch.

2.8 RELATED WORK

The work is motivated by the large expansion of modern cyber-physical systems (CPS) research into areas of machine intelligence [39, 40, 41] and autonomy to enable progressively broader categories of tomorrow's mission-critical applications [42]. Current machine learning software has been very successful at producing run-time inference algorithms that approach or exceed capabilities of human perception [43]. Of particular promise have been recent advances in neural networks [44, 45]. However, mainstream deep neural network inference algorithms are not designed explicitly with *timing and criticality constraints* of cyber-physical systems in mind, generating a need to refactor modern neural network software.

In the broader neural network research literature, much work was done on model compression and acceleration [46, 47, 48]. Examples include parameter quantization [49], edge pruning [50], node pruning [51], and dimensionality reduction (*e.g.*, factorization [52], sparsification [53], low-rank projection [54], or domain transform [55]), as well as combinations thereof [50]. We complement that work by introducing the notion of prioritization into the AI workflow. We exploit physical aspects of the platform and the application to enable additional reductions in cost while improving predictability, and timeliness. We expect that this improvement will significantly alter the price-capability trade-off of intelligent real-time embedded systems, making a new range of applications possible with increased autonomy

at a lower cost.

Recent efforts on AI-empowered real-time systems addressed CPU/GPU scheduling for pipelined machine inference [56, 57, 58, 59, 60, 61, 62], machine-learning library optimization [30], resource and energy management [63, 64, 65], and communication and collaboration protocol design [66, 67, 68, 69, 70, 71, 72]. Several novel cyber-physical applications with deep learning were introduced [73, 74, 75, 76, 77, 78]. Autonomous driving emerged as a flagship application motivating AI-empowered real-time system design [79]. Extensive hardware and software evaluations have been performed to understand its real time performance [3, 80, 81, 82]. Recent papers refactored deep neural networks to satisfy dynamic execution-time constraints during inference [30, 83, 84, 85, 86]. For example, Bateni *et al.* [83] applied a combination of different layer-wise network approximation techniques to meet target deadlines. Lee *et al.* [85] introduced dynamic subnetwork construction for DNNs (where the subnetwork with best performance that meets time constraints is selected at runtime). Heo *et al.* [86] proposed multi-path neural networks for real-time object detection systems. Similarly, they dynamically change the DNN's execution path to meet deadlines. However, all these efforts are limited to configuring neural network execution for frame by frame processing. In contrast, we break-up individual frames into regions of different degree of criticality and process such regions in priority order, as opposed to the strict frame arrival (FIFO) order to mitigate algorithmic priority inversion.

2.9 LIMITATIONS AND DISCUSSION

However, there are also limitations in our external-cueing based attention scheduling framework. First, since the imprecise computation models for neural networks only work with classification networks, we assume a perfect localization by the LiDAR clustering, which does not hold in practice. Therefore, how to define a similar imprecise computation paradigm for general neural networks is an important problem we need to investigate in the future, such that the deployed visual DNNs can not only localize the appeared objects but also classify their categories with different levels of confidence when different number of stages are executed. Second, our method on calculating the relative-velocity can be noisy because of the noise amplification caused by the distance differentiation operations. Alternative approaches include using integration-based velocity measurement or Radar-based velocity measurement with Doppler effect. Third, the assumption on the criticality design is still simple and can not handle complicated realworld scenarios. We still need to take more practical considerations (*e.g.*, driving scenarios and traffic regulations) into the attention design.

CHAPTER 3: ATTENTION-BASED SCHEDULING WITH SELF-CUEING

3.1 OVERVIEW

This work introduces a self-cueing real-time attention scheduling framework for machine perception in cyber-physical systems that minimizes a notion of system uncertainty. Modern machine perception pipelines rely on neural networks, such as YOLO [87], to perform object detection, localization and classification tasks (thereafter collectively called *detection tasks* for short, where no ambiguity arises), and feed downstream components such as navigation control. Attention scheduling refers to reducing the area inspected by the detection neural network in some criticality-dependent fashion to improve perception pipeline efficiency. It mimics the allocation of human cognitive capacity to focus on elements that matter most in a complex scene, as opposed to giving all elements of the scene the same level of attention. Unlike previous work that relied on an external ranging sensor to cue attention [38], in this work, we do it by exploiting *temporal correlations* in video streams. Specifically, we preferentially inspect those parts of the scene that are more important and *change less predictably* from frame to frame. We formulate this problem as one of minimizing maximum weighted (location) uncertainty, and develop a near-optimal real-time scheduling algorithm to schedule the selected regions for processing (by the perception neural network) on the GPU. Autonomous driving is used as an example application, although the design generalizes to other cyber-physical applications as well.

The work is motivated by the increasing popularity of *visual machine perception* (*i.e.*, the process of extracting relevant knowledge of immediate surroundings from camera images) as a foundation for a wide spectrum of intelligent cyber-physical applications [88, 89, 90]. Advances in deep neural networks have significantly improved perception quality of many vision tasks, such as image recognition [19], object detection [91, 92], and semantic segmentation [93]. However, delivering real-time results by running computationally intensive neural network models on resource-limited embedded platforms has remained a key challenge. Additionally, as mentioned by Huang *et al.* [94], objects in driving scenarios are about three times smaller than objects in general detection scenarios. Thus, high image resolutions are needed during inference to achieve sufficient detection quality to ensure self-driving safety.

Many existing efforts on enabling real-time neural network inference on embedded platforms focus on neural network compression [48, 49] and cloud offloading [95, 96]. There are a couple of challenges with these approaches. On one hand, in compression, detection quality of all parts of the scene is affected with no regard to criticality. On the other hand, offloading

(part of) the computation to the cloud requires a stable and fast network connection, which is not guaranteed in many driving scenarios. As an alternative approach, Liu *et al.* [38] proposed to slice input images into regions of different criticality, so that inspection of critical regions can be prioritized. The work relies on an external ranging sensor (*e.g.*, LiDAR) to determine which parts of the scene are more critical to inspect first (*e.g.*, closer objects or more quickly approaching objects). Unfortunately, LiDAR is not available on all platforms (*e.g.*, Tesla has famously opposed using it). Furthermore, reliance on multiple sensors increases cost and requires precise calibration and synchronization, where the degradation of either could cause downstream detection issues. By introducing a self-cueing attention scheduling framework that works with the camera alone without relying on external sensor inputs, we side-step the above fusion challenges.

Our scheduler runs full-frame detections at larger time intervals (*e.g.*, every 1 or 2 seconds). Between full-frame detections, we exploit optical flow-based object tracking [97] to predict object locations over time. Intuitively, if we can estimate the location of an object in a new frame with high confidence from its past trajectory, there is no need to look at it again. In the absence of new observations, however, uncertainty in object locations[1] increases over time. Selected parts of new frames are therefore re-inspected by the perception system to localize selected objects and keep overall uncertainty bounded between full-frame detections. We call such selective inspection *partial-frame detections*. We define the time interval between two full-frame detections as a *scheduling horizon*, and propose a scheduling algorithm to decide the schedule of partial-frame detections within each horizon to minimize a notion of *maximum weighted uncertainty*.

We implement the proposed scheduling framework on an NVIDIA Jetson Xavier board, and empirically evaluate its performance using real world driving datasets. The results show that the proposed policy achieves high detection, localization, and classification quality for both regular objects and critical objects under different workloads (compared to full-frame detection without resource constraints). It also provides better responses to physically close objects in our evaluation.

3.2 SYSTEM ARCHITECTURE

Consider an autonomous system with a camera that continuously observes the surrounding environment at a fixed frame rate, and an object detector (*e.g.*, YOLO) that is able to automatically localize and categorize all appeared objects in the captured images. The

[1]We interchangebly use uncertainty to denote location uncertainty later.

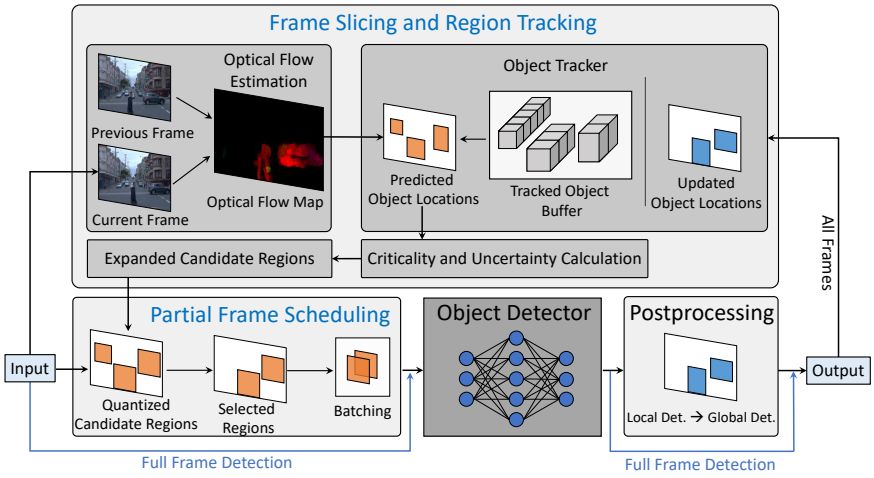


Figure 3.1: Overview of the proposed framework. Since we make no assumption or modification on the deployed object detection network, the proposed framework is generally applicable to any object detector.

detector can accept input images of sizes chosen from a discrete set of selectable options. The deployed detector is typically computationally intensive such that detection on a full image can not finish in real time (*i.e.*, before the next frame arrives). Instead, we process full frames at a longer interval T (say, 1-2 seconds). We refer to such processing as *full-frame detections*. Between them, we identify regions of interest in intermediate frames, and pass them to the detector, which are called *partial-frame detections*. The problem addressed in this work is: What regions of each intermediate frame should be passed to the detector under limited computation resources and time budget? Two core components are thus included in the proposed architecture: (i) the *frame slicing and region tracking module*, and (ii) the *partial-frame scheduling module*. The former extracts the parts of each frame as candidates, then the latter selectively schedules some parts to be inspected by the detector. An overview of our architecture is shown in Figure 6.1.

Frame Slicing and Region Tracking: This module slices image frames between full-frame detections into regions where objects may present. After a full-frame detection localizes all appeared objects, an optical-flow based tracking algorithm tracks the object locations in the following frames (until the next full-frame detection) based on the estimated pixel motions. At each frame, the module slices the partial regions around the tracked objects, considering the uncertainty in their predicted locations. Backgrounds, like the sky, are filtered out and will not be processed by the detector. The sliced partial regions, are selectively inspected by the detector, after which we are able to obtain the exact locations of the corresponding objects again, so their location uncertainty is reset.

Partial Frame Scheduling: This module decides the subset of sliced regions to pass

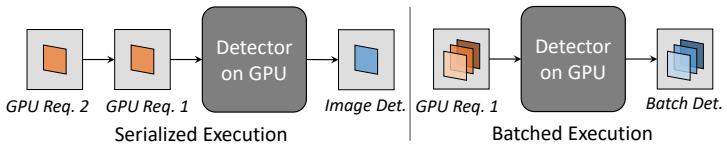


Figure 3.2: Comparison between **serialized execution** and **batched execution**.

to the detector at each frame, in order to maintain a low uncertainty on the location of each tracked object. Every object is associated with a criticality indicating its application-level importance, and a location uncertainty curve in the tracking algorithm. Uncertainty increases monotonically (at possibly different rates) with time until the next detection. The partial-frame detections of objects with low criticality or slow uncertainty growth are skipped at some frames. In addition, when making decisions, the scheduling algorithm also considers *task batching* on modern GPUs, which means multiple regions can be batched together and submitted as a single GPU request (as shown in Figure 3.2), *as long as they have the same size* (because low-end GPUs can only batch identical computational kernels). Batched processing can achieve much lower latency than serialized processing.

3.3 FRAME SLICING AND REGION TRACKING

In this section, we first provide background on optical flow, then introduce our flow-based object tracking algorithm, finally explain how location uncertainties are calculated.

3.3.1 Optical Flow Background

Optical flow algorithms take two consecutive frames as input and estimate the pixel-level motion vectors between them, as caused by the relative movement between objects and the observer. The (dense) optical flow algorithms return a map of single pixel motions, which is called *optical flow map*. The RGB image is first converted to gray scale, where each pixel value represents the amount of light, i.e., intensity, at that location. We use $I(x, y, t)$ to denote the image intensity (i.e., value) at pixel (x, y) of frame t. The optical flow map is a matrix of coordinate displacements (d_x, d_y), such that:

$$I(x, y, t) = I(x + d_x, y + d_y, t + 1). \tag{3.1}$$

Optical flow assumes that the pixel intensities of an object are constant across two consecutive frames. Compared to the block matching algorithms [98] used in video encoders for motion estimation that restrict the search area to a limited range, optical flow is more

accurate in long range motion estimation. In this work, we use the DIS method [97], which is a widely used and efficient dense flow estimation algorithm. Neural optical flow models, like FlowNet [99], though more accurate, are too resource consuming and not considered.

3.3.2 Optical Flow-based Object Tracking

Our optical flow-based tracking algorithm tracks locations of objects between two full-frame detections. It uses optical flow as a non-parametric motion model, which outperforms the parametric motion models in conventional tracking algorithms, *e.g.*, Kalman Filter in SORT [33], with more accurate motion estimation, when the object is only previously seen once and where the intermediate detections are skipped. This is because optical flow accepts any previous frame to calculate an up-to-date motion. Instead, parametric models often require a sequence of past locations before convergence.

Algorithm 3.1 details our tracking algorithm. We start from the set of objects received from the last full-frame detection, and track their locations until the next full-frame detection. Each time a new frame arrives, we first compute its optical flow map compared to the last frame, and correspondingly calculate the following three regions for each object:

① **Predicted Object Location:** It tightly bounds the most likely (predicted) object location from the optical flow map. We assume the object size does not change between two detections. If a partial-frame detection task is *not* scheduled, we use this location as a best guess of current object location. Otherwise, we update to the actual detected location (and size).

② **Expanded Candidate Region:** It expands the predicted location on account of uncertainty. This is the area that should be inspected by the detector if we want to localize the object again. It is a box whose area keeps expanding as long as no partial-frame detection task is scheduled.

③ **Quantized Candidate Region:** We pad the expanded candidate region to the nearest quantized size from a preset set to facilitate task batching, because only regions with the same size can be batched.

Figure 3.3 illustrates the difference between the three regions. Next, we explain how they are calculated step by step.

Computing Predicted Object Locations: To compute the predicted location for an object, we need to derive a representative motion vector for it based on the pixel-wise optical flow map. We choose the median motion vector among all pixels that lie within the previous object bounding box to represent the holistic object motion. The median motion is chosen over the mean motion to eliminate the impact of static background pixels (*e.g.*, road or sky).

Algorithm 3.1: Optical Flow-based Object Tracking

Input: Set of object $\{1, \ldots, N\}$, a sequence of $K-1$ frames between two full detections, object detector.

Maintain a set of object tracks for target objects;
for *frame* $k = 2, \ldots, K$ **do**
  Calculate the flow map between frame k and $k-1$;
  for *object* $i = 1, \ldots, N$ **do**
    Calculate object representative flow $(dx_i^{(k)}, dy_i^{(k)})$ by taking the median flow of previous object location;
    Update tracked object center location $\tilde{cx} := cx_i^{(k-1)} + dx_i^{(k)}$, $\tilde{cy} := cy_i^{(k-1)} + dy_i^{(k)}$;
  end
  Generate set of partial detections by the object detector;
  Data association using Hungarian algorithm between object tracks and new detections using IoU metric;
  for *object* $i = 1, \ldots, N$ **do**
    if *mapped with a new detection* **then**
      new object location := mapped detection location
    end
    else
      new object location := predicted object location
    end
  end
end

Computing Expanded Candidate Regions: This region accounts for uncertainty in object locations, and gives where we can find the object at every frame. It starts from the detected object location, and then keeps expanding, until a new detection arrives. Specifically, at a new frame k, we use $[x_{min}^{(k)}, y_{min}^{(k)}, x_{max}^{(k)}, y_{max}^{(k)}]$ to denote the new object location, and use $\mathbf{D} = [\mathbf{D}_{\hat{x}}, \mathbf{D}_{\hat{y}}]$ to denote the partial flow matrix corresponding to its previous expanded candidate region, say $[\hat{x}_{min}^{(k-1)}, \hat{y}_{min}^{(k-1)}, \hat{x}_{max}^{(k-1)}, \hat{y}_{max}^{(k-1)}]$. If the previous expanded candidate region completely covers the previous object location, then the new object location satisfies:

$$\hat{x}_{min}^{(k-1)} + \min_{d_{\hat{x}} \in \mathbf{D}_{\hat{x}}} d_{\hat{x}} \leq x_{min}^{(k)} \leq x_{max}^{(k)} \leq \hat{x}_{max}^{(k-1)} + \max_{d_{\hat{x}} \in \mathbf{D}_{\hat{x}}} d_{\hat{x}}, \tag{3.2}$$

$$\hat{y}_{min}^{(k-1)} + \min_{d_{\hat{y}} \in \mathbf{D}_{\hat{y}}} d_{\hat{y}} \leq y_{min}^{(k)} \leq y_{max}^{(k)} \leq \hat{y}_{max}^{(k-1)} + \max_{d_{\hat{y}} \in \mathbf{D}_{\hat{y}}} d_{\hat{y}}. \tag{3.3}$$

Thus, we define the new expanded candidate region as:

$$[\hat{x}_{min}^{(k-1)} + \min_{d_{\hat{x}} \in \mathbf{D}_{\hat{x}}} d_{\hat{x}},\ \hat{y}_{min}^{(k-1)} + \min_{d_{\hat{y}} \in \mathbf{D}_{\hat{y}}} d_{\hat{y}},\ \hat{x}_{max}^{(k-1)} + \max_{d_{\hat{x}} \in \mathbf{D}_{\hat{x}}} d_{\hat{x}},\ \hat{y}_{max}^{(k-1)} + \max_{d_{\hat{y}} \in \mathbf{D}_{\hat{y}}} d_{\hat{y}}]. \tag{3.4}$$

The expanded candidate region will also completely cover the object location in the new

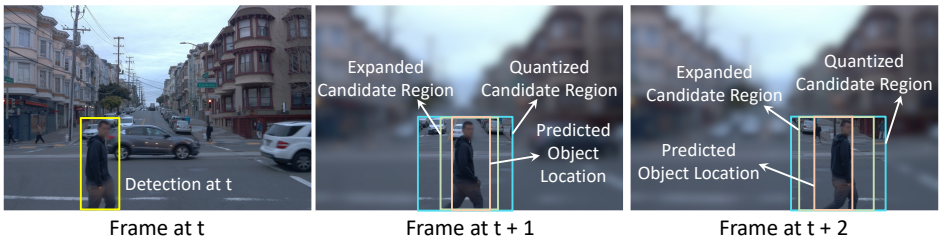


Figure 3.3: Comparison between different region concepts. **Predicted Object Location:** The predicted object location according to estimated flow map. **Expanded Candidate Region:** The expanded region including uncertainty in object location. **Quantized Candidate Region:** We pad the expanded candidate region to the nearest quantized size to facilitate task batching.

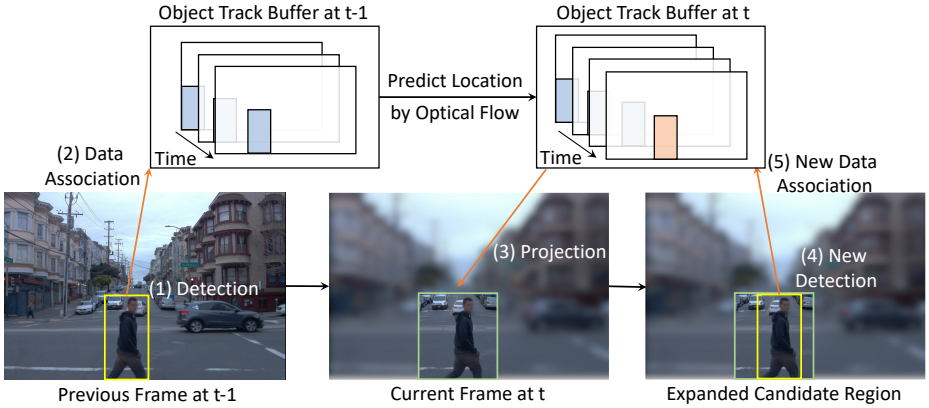


Figure 3.4: Relationship between detection, tracking, and projection.

frame. Since the expanded candidate region starts from the exact object location, it holds by induction that the expanded candidate region covers the (groundtruth) object location at every future frame, even when detections are skipped at some intermediate frames.

Computing Quantized Candidate Regions: Recall that task batching on (low-end) GPUs requires the batched images to have the same size and leads to much higher time efficiency. To facilitate task batching, we pad the expanded candidate region to the nearest quantized *target size* s_i chosen from a finite set, $s_i \in \{s_1, \ldots, s_M\}$. The padded region is then called the *quantized candidate region*. We assign a fixed target size s_i to each object within a scheduling horizons. We provide two justifications for this choice: First, the quantized size is larger than the initial object size, so it leaves space for object size increase in upcoming frames. Second, if the expanded candidate region expands beyond s_i, we reduce its resolution to make it fit into s_i. As we will show later, downsizing large objects does not degrade the detection quality.

Data Association: After we receive the set of detected object locations from the detector, we perform data associations between the existing object tracks (represented by their

predicted object locations) and the new detected bounding boxes. We do so by using the Hungarian algorithm based on their location overlaps with an Intersection-over-Union (IoU) metric. We then update the mapped object locations to the newly detected locations. Those objects not inspected by the detector in a given frame will retain their predicted object locations. A graphical illustration of the tracking process is provided in Figure 3.4.

3.3.3 Object Location Uncertainty

The *object location uncertainty* reflects our confidence on the predicted object location, compared to the current actual object location. Intuitively, if the size of the expanded candidate region (i.e., area where the object may appear) is close to the predicted object location, we have a low uncertainty (i.e., high confidence) on the predicted object location; otherwise, if the object can appear at much larger area than the predicted object location, we have a high uncertainty (i.e., low confidence) on the predicted object location.

We assign an *object weight* $w_i = v_i \cdot u_i$ to each tracked object $\mathcal{O}_i$, which is the product of two factors: 1) The *object criticality* v_i representing the application-specific importance, which is static within a scheduling horizon. Our algorithm is agnostic to the of object criticality, so we leave it for future explorations. 2) The *uncertainty growth rate* u_i, defined as the average amount of its candidate region expansion speed. The uncertainty for object $\mathcal{O}_i$ is reset to wt_d after detection, where t_d is the detection time which can be either full-frame or partial-frame latency. The reset value of uncertainty is caused by the time between when the image frame is sampled, and when we produce its detections. When the first frame after the full-frame detection arrives, we calculate the uncertainty growth rate as $u_i = \sqrt{S_i^{ECR}/S_i^D - 1}/t_f$, where S^{ECR} is the area of the expanded candidate region, S_i^D is the area of the last detected location, and t_f is the full-frame detection latency. The uncertainty grows linearly with time if no new detection is performed, because the expanded candidate region keeps expanding while the predicted object size is constant. By separating the uncertainty into the weight factor and the elapsed time, we can simply denote the weighted uncertainty of object $\mathcal{O}_i$ as $U_i(t) = w_i(t - t_i)$, where $t - t_i$ is the elapsed time since the start of its last detection t_i. The object weight w_i, as explained later, will decide the detection frequency of object $\mathcal{O}_i$ in the proposed partial-frame scheduling algorithm.

3.4 PARTIAL FRAME SCHEDULING

In this section, we first explain the detector execution model and formulate the partial-frame scheduling problem, then propose an algorithm that produces near-optimal schedules.

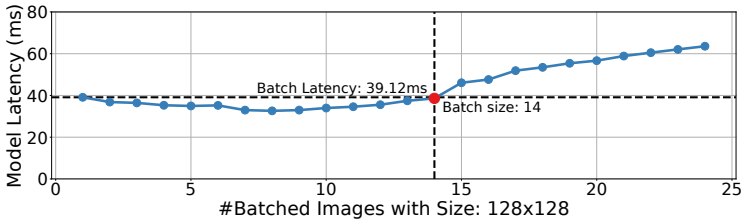


Figure 3.5: YOLO execution latencies of 128 × 128 images with different batch sizes on Jetson Xavier. The inflection point is highlighted in red, where the batch size is 14. The batch execution latency is set as the maximum latency before the inflection point, which is 39.12ms in this curve.

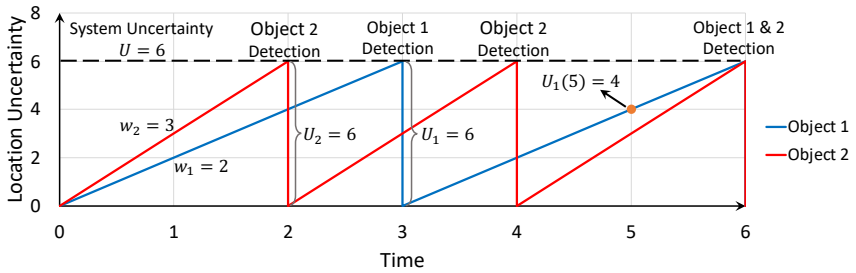


Figure 3.6: Graphical illustration on uncertainty definitions and why to set the object detection frequency proportional to their object weights. For simplicity of illustration, we ignore the uncertainty caused by detection latency, since it is typically much smaller than the intervals between detection tasks.

3.4.1 Object Detection Model

We divide the time into fixed-length segments, where each segment is called a *scheduling horizon* T. The autonomous platform is equipped with a single GPU that runs the detector. We run a full-frame detection at the first frame of each scheduling horizon, which identifies N objects $\{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_N\}$. K frames are captured between two full-frame detections. We subtract the latency of the preprocessing steps (such as tracking, slicing and batching) and the full-frame detection from T to get the time budget T' of partial-frame detections. Each object $\mathcal{O}_i$ is associated with a target size $s_i \in \{s_1, \ldots, s_M\}$, within horizon T, which restricts the size of its quantized candidate regions and facilitates batching. For each target size s, there exists a maximum number of regions that can be batched and processed in parallel on the GPU. We call it the *batching limit* κ_s for size s. Although the detector execution time can increase with the number of batched regions, by appropriately setting the batching limit, we operate in a region where execution time changes only slightly with batching (before an inflection point is reached where the slope increases, as shown in Figure 3.5). We denote the worst-case batch execution time by τ_s. In other words, the GPU can simultaneously run partial-frame detections for κ $(1 \leq \kappa \leq \kappa_s)$ objects of target size s within time τ_s.

3.4.2 Scheduling Problem Formulation

A good tracking system should selectively run partial-frame detections to maintain low location uncertainty on each object throughout the scheduling horizon. Recall that the (weighted) location uncertainty of object $\mathcal{O}_i$ at time t is $U_i(t) = w_i(t - t_i)$, where t_i is the last detection time. Without loss of generality, we assume $w_1 \leq \ldots \leq w_N$. The maximum uncertainty for object $\mathcal{O}_i$ over the scheduling horizon is denoted by $U_i = \max_{t\in[0,T']} U_i(t)$, where we define $t = 0$ as the time when the full-frame detection finishes. Our goal is to minimize the maximum weighted uncertainty over all objects, which we refer to as the *system uncertainty* U. It is defined as $U = \max_{i\in\{1,\ldots,N\}} U_i$. These concepts are visually illustrated in Figure 3.6. The problem we study, is to design a *schedule* of partial-frame detections such that the system uncertainty is minimized. A schedule specifies the ordering and batching of partial-frame detections. It is formally defined as follows.

Definition 3.1 (Schedule)**.** A schedule is a sequence of tuples $(\mathcal{N}^1, s^1, t^1, k^1), (\mathcal{N}^2, s^2, t^2, k^2)$, $\ldots, (\mathcal{N}^I, s^I, t^I, k^I)$. Both $t^1, \ldots, t^I$ and $k^1, \ldots, k^I$ are in non-decreasing order. For a generic j-th tuple, it represents the j-th batch, where:

- $\mathcal{N}^j$ is the subset of objects that get detected in the batch. No object can appear more than once in the subset.
- s^j denotes the target size of the batch.
- $t^j \in [0, T']$ is the start execution time of the batch.
- $k^j \in \{1, \ldots, K\}$ represents the frame on which the partial-frame detection is run.

A schedule is *feasible* if it satisfies for each batch j: (1) The number of batched regions is within the batching limit, *i.e.*, $|\mathcal{N}_j| \leq \kappa_{s^j}$. (2) We define the *valid period* of the frame k as the interval between its arrival and before the next frame arrives, and the batch can only run on the currently valid frame. (3) The start time of the batch is no earlier than the finish time of the previous batch, *i.e.*, $t^j \geq t^{j-1} + \tau_{s^{j-1}}$. (4) The finish time of the last batch is no later than T', *i.e.*, $t^I + \tau_{s^I} \leq T'$.

Note that each feasible schedule can be executed on the physical machine, and each execution on the physical machine can be translated to a feasible schedule. With the above preliminaries, we formulate our problem as follows.

Definition 3.2 (Partial-Frame Detection Scheduling Problem)**.** The Partial-Frame Detection Scheduling (PFDS) problem asks for a feasible schedule that minimizes the system uncertainty within the time budget T' of a scheduling horizon.

The PFDS problem requires us to intelligently select subsets of objects to run and batch on each frame. Although the PFDS problem can be optimally solved by the dynamic-programming paradigm. However, the resulted computational complexity would be high. Instead, we will propose a low-complexity policy, called the Batched Proportional Balancing (BPB) policy, that computes approximately optimal schedules with provable uncertainty guarantee.

3.4.3 Scheduling Policy

The general idea of the proposed **Batched Proportional Balancing (BPB)** policy is to set the number of partial-frame detection tasks[2] for each object proportional to its object weight, such that the objects with high criticality or high uncertainty growth would receive more attention (*i.e.*, computation resource). For each object $\mathcal{O}_i$, we use the *detection frequency* x_i to denote its number of scheduled partial-frame detection tasks within the scheduling horizon. The *detection frequency set*, is thus defined as:

Definition 3.3 (Detection Frequency Set)**.** The detection frequency set $\{x_1, \ldots, x_N\}$ is a set of detection frequencies corresponding to the number of partial-frame detection tasks of all objects in the scheduling horizon where, for each object $\mathcal{O}_i$, x_i partial-frame detection tasks are scheduled.

We aim at computing a detection frequency test where the detection frequency x_i for object $\mathcal{O}_i$ is *approximately proportional* to its weight w_i (*i.e.*, `Proportional`), and make sure the intervals between consecutive partial-frame detections of each object are evenly distributed in the schedule (i.e. `Balancing`), as shown in Figure 3.6. The design so far seems similar to the well-studied pinwheel scheduling problem [100]. However, we go a step further by considering **task batching** (*i.e.*, `Batched`), where we need to simultaneously decide *when to detect each object* and *how to batch the detections of objects* such that the system uncertainty is minimized. Improper batching may result in low utilization on the GPU and much higher system uncertainty. The pseudocode of the BPB policy is presented in **Algorithm 3.2**. It searches for a detection frequency set with the minimum system uncertainty, and invokes the **Batch-Aware Scheduling (BAS)** algorithm (**Algorithm 3.3**) as a sub-procedure to derive an optimal schedule for a given detection frequency set.

To reduce the search effort, the BPB policy first proportionally derives the *normalized detection frequencies* of objects such that the object with the smallest weight is detected only once. They are computed by dividing the object weights by the minimum weight, and

[2]We also use *a task* to denote a partial frame detection task for an object.

Algorithm 3.2: The BPB Policy

Input: Object set $\{\mathcal{O}_1, \ldots, \mathcal{O}_N\}$, weights $\{w_1, \ldots, w_N\}$, number of frames K for partial-frame detections.

Output: A feasible schedule with minimized uncertainty.

Sort and reindex the objects such that $w_1 \geq \ldots \geq w_N$;
for $i = 1, \ldots, N$ **do**
  $x_i := 2^{\lfloor \log_2(w_i/w_N) \rfloor}$;
end
$\mathcal{C} = \{\frac{1}{x_1}, \frac{1}{x_2}, \ldots, 1, 2, 3, \ldots, \lfloor \frac{K}{x_1} \rfloor, \frac{K}{x_1}\}$;
Binary search for the maximum $c \in \mathcal{C}$ such that the schedule computed by **Algorithm 3.3** for task set $\{\lfloor cx_1 \rfloor, \ldots, \lfloor cx_N \rfloor\}$ is feasible (*i.e.*, the finishing time is no larger than T');
Return the schedule for the task set of the maximum c.

rounding down to the nearest power of 2 if they are not[3]. Let the normalized detection frequency set be $\{x_1, \ldots, x_N\}$. BPB then searches a maximum scaling factor c such that the schedule returned by the BAS algorithm for the detection frequency set $\{\lfloor cx_1 \rfloor, \ldots, \lfloor cx_N \rfloor\}$ is feasible. Note that the scaling factor c can be smaller than one, and thus in the resulting detection frequency set, $\lfloor cx_n \rfloor$ can be zero for some objects. Such objects will not be scheduled. As we will show in the sequel, if the schedule calculated by the BAS algorithm for c is feasible, so is the schedule calculated by the BAS for any $c' \leq c$. Thus, the maximum c can be identified via binary search due to this monotonicity property.

The Batch-Aware Scheduling (BAS) algorithm (**Algorithm 3.3**) computes an optimal schedule that minimizes the system uncertainty for a given detection frequency set $\{x_1, \ldots, x_N\}$. BAS works as a two-step procedure. First, BAS maps the partial-frame detection tasks for objects to $L = x_N$ temporally distributed virtual bins $\{B_1, \ldots, B_L\}$. The virtual bins do not correspond to camera frames. No object can have more than K partial-frame detections in a scheduling horizon, so we assume $L \leq K$. BAS sequentially assigns the tasks of each object $\mathcal{O}_i$ in decreasing order of x_i. Since each x_i is an integer power of 2 multiple of the minimum non-zero element in $\mathcal{C}$, when mapping tasks for object $\mathcal{O}_i$, BAS only designates the mapping of its first task to the first L/x_i bins[4] and replicates the mapping for the remaining tasks to the corresponding bins in remaining subsets. By doing so, when assigning tasks of an object, the matched bins in different subsets always have perfectly symmetric load. The first task of each object is assigned in a batch-aware load-balanced fashion. At object $\mathcal{O}_i$, BAS first checks whether there is a bin that has incomplete batch with size s_i , *i.e.*, the number of tasks with size s_i in the bin is not a multiple of κ_{s_i}. If such a bin exists, it assigns the

[3]This operation is used to align the detection times among objects to trigger more batching opportunities.

[4]L/x_i is an integer since both L and x_n are powers of 2 multiples of the minimum non-zero element in $\mathcal{C}$ and $x_n \leq L$.

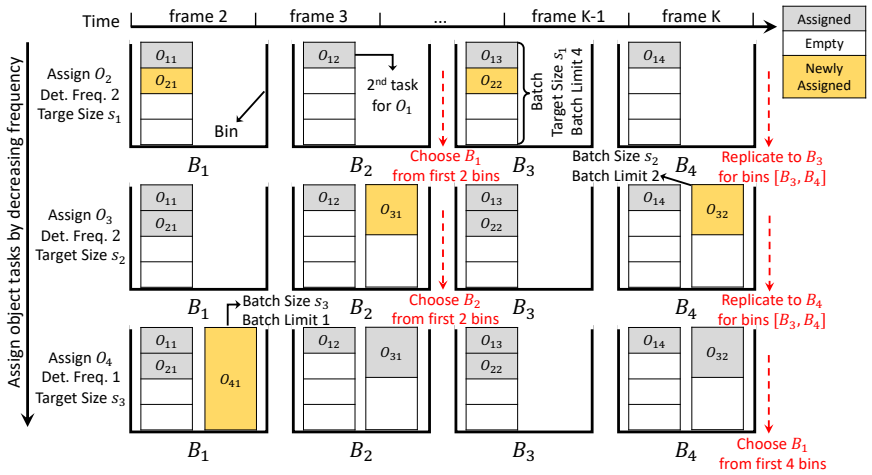


Figure 3.7: Graphical illustration on how BAS generates the task-bin mapping. We have four objects denoted by (object, detection frequency, target size): $(\mathcal{O}_1, 4, s_1)$, $(\mathcal{O}_2, 2, s_1)$, $(\mathcal{O}_3, 2, s_2)$, $(\mathcal{O}_4, 1, s_3)$. We have 4 (virtual) bins, which are **not** aligned with the frame boundaries. For object $\mathcal{O}_2$, its first task is assigned to bin B_1 because there is an incomplete batch with size s_1, and the decision is replicated to bin B_3. For object $\mathcal{O}_3$, its first task is assigned to bin B_2, and the decision is replicated to bin B_4. The task for object $\mathcal{O}_4$ is assigned to bin B_1 with the min load.

task to that bin; otherwise, it assigns the task to the bin with the minimum load. The *bin load* is defined as the total execution time for batches in the bin. The assignment process is visually illustrated in Figure 3.7. Second, it converts the generated task-bin mapping to a schedule by sequentially executing the bins, and greedily batching tasks with the same target size in each bin. When compositing a batch, we select the valid frame at that time to run the partial-frame detections.

3.4.4 Theoretical Analysis

In this part, we analyze the approximation ratio on achieved system uncertainty of our BPB policy. Let U^* be the minimum system uncertainty under any feasible schedule. In Theorem 3.1, we establish that the BPB policy computes schedules with system uncertainty that is at most $3\times$ the optimal if the object weights are powers of 2, or at most $5\times$ the optimal under general object weights.

Theorem 3.1. Let U_{BPB} be the system uncertainty under the BPB policy. If the object weights $w_1, \ldots, w_N$ are all integer powers of 2, then $U_{BPB} \leq 3U^*$, otherwise in general case, $U_{BPB} \leq 5U^*$.

Next we formally prove this theorem. We first reindex the objects in the decreasing order of their weight factors, *i.e.*, $w_1 \geq \cdots \geq w_N$. As shown in Figure 3.8, the overall object

Algorithm 3.3: Batching-Aware Scheduling

Input: Detection frequency set $\{x_1, \ldots, x_N\}$
Output: A schedule for the detection frequency set
// (1) Calculate the task-bin mapping.
$L := x_1$;
for $i \in \{1, 2, \ldots, N\}$ *(decreasing order of* x_i*)* **do**
  Let L_i be the first L/x_i bins $\{B_1, \ldots, B_{L/x_i}\}$;
  $s_i :=$ the target size of $\mathcal{O}_i$;
  if $\exists B_l \in L_i$ *with incomplete batch of size* s_i **then**
    Add the first task of $\mathcal{O}_i$ to B_l;
  end
  else
    Add the first task of $\mathcal{O}_i$ to the bin in L_i with the minimum load;
  end
  Replicate the mapping of the remaining tasks of n to the remaining subset of bins;
end
// (2) Convert the task-bin mapping to a schedule.
$j = 1, t^j = 0$, schedule $\mathcal{S} = \emptyset$;
for $l \in \{1, \ldots, L\}$ **do**
  $t_j := \max\{t_j, \text{start of valid period of the } l\text{-th frame}\}$.
  for $s \in \{s_1, \ldots, s_M\}$ **do**
    $\kappa :=$ the number of objects of size s in B_l;
    while $\kappa > 0$ **do**
      $\mathcal{N}^j := \min\{\kappa, \kappa_s\}$ objects of size s in B_l;
      $k :=$ the most recent camera frame at t_j
      Add $(\mathcal{N}^j, s, t^j, k)$ to $\mathcal{S}$;
      $t^{j+1} := t^j + \tau_s$, $j := j + 1$;
      Remove the selected objects from B_l;
    end
  end
end
Return the schedule $\mathcal{S}$

uncertainty comes from two parts: *interval between detection tasks* and *detection latency*[5]. We consider the uncertainty caused by detection latency because after we finish detection task on a frame, the obtained object locations do not perfectly correspond to the current time, but the time when the input frame is sampled. This part of uncertainty is amplified on full-frame detections. We first utilize the symmetric structure of the scheduled computed by BAS (*i.e.*, the mapping of each subsequent task of an object is a duplicate of the first task to the corresponding subset of bins), to bound the uncertainty caused by intervals between

[5]Here we ignore the delay between when the image frame is captured and when it is used for detection. Since we always use the newest frame for detection, this delay is bounded by the camera sampling period.

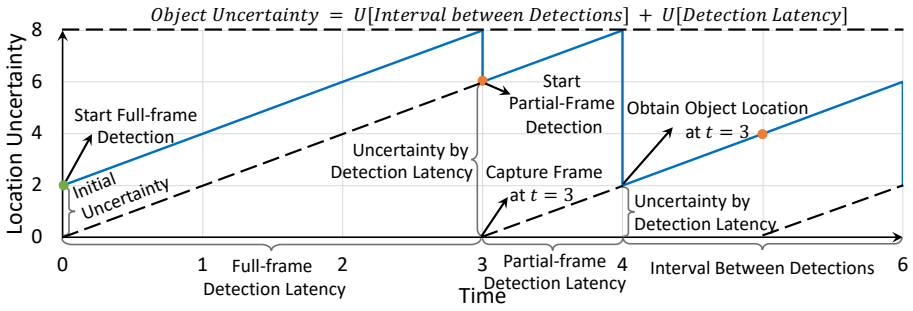


Figure 3.8: The object uncertainty comes from both the *detection latency* and *interval between detections*. After each detection task, the uncertainty is reset to the value caused by only detection latency.

detection tasks. Then, we include the uncertainty caused by detection latency, and derive the bound for overall uncertainty. The proof procedure consists of the following steps:

- **Step 1:** We prove that the load difference $\bar{\lambda}_l(i) - \underline{\lambda}_l(i)$, between the maximum bin load $\bar{\lambda}_l(i) := \max_l \lambda_l(i)$ and the minimum bin load $\underline{\lambda}_l(i) := \min_l \lambda_l(i)$, is always bounded, where $\lambda_l(i)$ is the load for bin B_l after assigning the first i objects.

- **Step 2:** We prove that, given a detection frequency set, BAS is optimal in minimizing the system load, *i.e.*, , the sum of load on all bins $\sum_l \lambda_l$.

- **Step 3:** We prove the bound on the system uncertainty caused by intervals between detections, by bounding the maximum bin load with the optimal system uncertainty.

- **Step 4:** We include the uncertainty caused by detection latency, and prove the bound on the overall system uncertainty.

Next, we prove the first three steps as lemmas one by one, before proving the overall system uncertainty bound in the last step.

Step 1: We first prove that the bin load difference is bounded at every step of BAS.

Lemma 3.1. For each i, $\bar{\lambda}(i) - \underline{\lambda}(i) \leq \max\{\bar{\lambda}(i-1) - \underline{\lambda}(i-1), \tau_{s_i}\}$, where s_i is the target size for the object $\mathcal{O}_i$ and τ_{s_i} is its corresponding batch execution time.

Proof. For each i, let $L_i := \frac{L}{x_i}$, where x_i is the detection frequency for object $\mathcal{O}_i$. L_i is an integer power of 2 by construction. When assigning object $\mathcal{O}_i$, we have $\lambda_l(i) = \lambda_{l'}(i)$ for $l \equiv l' (\mod L_i)$ (as exemplified in Figure 3.7).

- If there exists an incomplete batch for target size s_i in first L_i bins, then $\bar{\lambda}(i) = \bar{\lambda}(i-1)$ and $\underline{\lambda}(i) = \underline{\lambda}(i-1)$, and the claim follows.

- If there is no incomplete batch, then based on induction, there are a subset of bins, that for all l such that $\lambda_l(n-1) = \underline{\lambda}(i-1)$, and they are equivalent modulo L_{i-1}. Since $L_{i-1} \leq L_i$ and $L_i \mod L_{i-1} \equiv 0$, there exists at least one bin $l \in L_i$ with load $\underline{\lambda}(i-1)$. After assigning $\mathcal{O}_i$ to l, its load will increase to $\underline{\lambda}(i-1)+\tau_{s_i}$. If $\underline{\lambda}(i-1)+\tau_{s_i} \geq \bar{\lambda}(i-1)$, we have $\bar{\lambda}(i) - \underline{\lambda}(i) \leq \tau_{s_i}$; otherwise, we have $\bar{\lambda}(i) - \underline{\lambda}(i) \leq \bar{\lambda}(i-1) - \underline{\lambda}(i-1)$.

QED.

Step 2: Next, we prove the optimality of the BAS algorithm in minimizing the system load of a given detection frequency set.

Lemma 3.2. Given a detection frequency set $\{x_1, x_2, \ldots, x_N\}$, we use λ_{BAS} to denote the total load of the schedule computed by the BAS algorithm (Algorithm 3.3). It minimizes the total load over all feasible schedules for the given detection frequency set, *i.e.*, $\lambda_{BAS} \leq \lambda$, with λ being the total load of any other feasible schedule.

Proof. As the tasks of different sizes cannot be batched together, the total load of a schedule is exactly determined by the batching composition of each size. For a given size, the total load of tasks of that size is minimized when its number of batches is minimized. It can be proven by induction that, following the decreasing order of x_i for all objects of the same size, BAS minimizes the number of batches for the first i objects. The base step is trivial, since no tasks of the same object can be batched. For the induction step, assume the claim holds for i. When deciding on the mapping of tasks on object $\mathcal{O}_{i+1}$, if there exists an incomplete batch, then the schedule by BAS has the same number of batches as that for the first i objects, from which the claim follows. If no incomplete batch exists, consider any partial schedule for the first i objects. If for that partial schedule, an incomplete batch exists, then it contains at least one more batch than the partial schedule by BAS for the first i objects. It is contradictory with the assumption that BAS minimizes the batch count for the first i objects. In both cases, for the first $i+1$ objects, the partial schedule by BAS contains no more batches than any other partial schedule, which completes the induction argument and the claim follows. QED.

Step 3: Then, we consider an optimal schedule that achieves the minimum system uncertainty $\tilde{U}^*$, without considering the uncertainty caused by detection latency, and prove the bound on system uncertainty achieved by BPB algorithm.

Lemma 3.3. Assume the object uncertainty is reset to 0 after each detection, if the object weights are all integer powers of 2, then $U_{BPB} \leq 2\tilde{U}^*$; otherwise in general case, $U_{BPB} \leq 4\tilde{U}^*$.

Proof. Let $\{\tilde{x}_1^*, \ldots, \tilde{x}_N^*\}$ be the set of detection frequencies executed under the optimal schedule. We construct its *feasible proper adaptation* following the procedure below.

- Under the optimal schedule, we have $U^* \geq \max_i \frac{w_i T'}{\tilde{x}_i^*}$. Let $\hat{i} = \arg\max_i \frac{w_i T}{\tilde{x}_i^*}$. For each i, $\frac{w_i}{\tilde{x}_i^*} \leq \frac{w_{\hat{i}}}{\tilde{x}_{\hat{i}}^*}$. Since each $\tilde{x}_i^*$ is an integer, it follows that $\tilde{x}_i^* \geq \frac{w_i \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \geq \lfloor \frac{w_i \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \rfloor$. We set $\hat{c} = \lfloor \frac{w_N \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \rfloor \geq 1$.[6]
- Let $x_i = 2^{\lfloor \log_2(w_i/w_N) \rfloor}$, that is, $\{x_1, \ldots, x_N\}$ is the output of step 3 of **Algorithm** 3.2, *i.e.*, the ratios of the detection frequency set of BPB. By definition, $x_N = 1$. According to the construction, for each i, we have $x_i = \frac{x_i}{x_N} = 2^{\lfloor \log_2(w_i/w_N) \rfloor} \leq \lfloor \frac{w_i}{w_N} \rfloor$.
- We define $\{\hat{c}x_1, \ldots, \hat{c}x_N\}$ as the proportional adaptation of the optimal detection frequency set. Since both $\hat{c}$ and x_i are both integers, we have $\lfloor \hat{c}x_i \rfloor = \hat{c}x_i$.

We next prove that the constructed proportional adaptation of the optimal detection frequency set is a feasible set that can finish within time budget T'. For each object $\mathcal{O}_i$, we have:

$$\hat{c}x_i \leq \hat{c} \lfloor \frac{w_i}{w_N} \rfloor = \lfloor \frac{w_N \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \rfloor \lfloor \frac{w_i}{w_N} \rfloor \leq \lfloor \frac{w_i \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \rfloor \leq \frac{w_i \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \leq \tilde{x}_i^* \tag{3.5}$$

Since the optimal schedule is feasible, there also exists a feasible schedule for the detection frequency set $\{\hat{c}x_1, \ldots, \hat{c}x_N\}$. Since we have proved in Lemma 3.2 that the BAS algorithm minimizes the system load, thus the factor c by BAS is greater than or equal to $\hat{c}$, *i.e.*, $c \geq \hat{c}$. In our BPB algorithm, for an object $\mathcal{O}_i$, its uncertainty is at most $w_i(\max_l \lambda_l)\frac{L}{x_i} = w_i(\max_l \lambda_l)\frac{x_1}{x_i}$. We define λ_l as the load (*i.e.*, total execution time of its batches) of bin B_l. We bound the system uncertainty by bounding the maximum bin load of the BPB schedule. We consider the following two cases.

(Case 1): If $\bar{\lambda}(N) \leq 2\underline{\lambda}(N)$, we have

$$\max_l \lambda_l = \bar{\lambda}(N) \leq 2\underline{\lambda}(N) = 2 \min_l \lambda_l \leq \frac{2\lambda_{BPB}}{L} \leq \frac{2T'}{cx_1} \leq \frac{2T'}{\hat{c}x_1} \tag{3.6}$$

From the construction of $\{x_1, \ldots, x_N\}$, we have for any object $\mathcal{O}_i$, $\frac{x_i}{x_N} \leq \frac{w_i}{w_N} \leq \frac{2x_i}{x_N}$, it follows that the weighted overall uncertainty of each object is upper bounded by

$$\frac{w_i x_1}{x_i} \cdot \max_l \lambda_l \leq \frac{w_i x_1}{x_i} \cdot \frac{2T'}{\hat{c}x_1} \leq \frac{4w_N T'}{\hat{c}x_N} = \frac{4w_N T'}{w_N \tilde{x}_{\hat{i}}^* / w_{\hat{i}}} \leq 4\tilde{U}^* \tag{3.7}$$

[6]Without loss of generality, we assume that $\lfloor \frac{w_N \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \rfloor \geq 1$ and $\frac{\tilde{x}_{\hat{i}}^*}{w_{\hat{i}}}$ is an integer; otherwise, we can just take the largest i with non-zero value of this equation and leave out the remaining objects.

(Case 2): If $\bar{\lambda}(N) > 2\underline{\lambda}(N)$, consider the last i where $\bar{\lambda}(i)$ increases (*i.e.*, $\bar{\lambda}(i) > \bar{\lambda}(i-1)$, we have $\bar{\lambda}(N) - \underline{\lambda}(N) \leq \bar{\lambda}(i) - \underline{\lambda}(i) \leq \tau_{s_i}$. We have $\tau_{s_i} \geq \max_l \frac{\lambda_l}{2}$. In this case, even under the optimal schedule, the maximum uncertainty of object $\mathcal{O}_1$ is at least $w_1\tau_{s_i} \leq \tilde{U}^*$, so we have $\max_l \lambda_l \leq \frac{2\tilde{U}^*}{w_1}$. Hence,

$$\frac{w_i x_1}{x_i} \cdot \max_l \lambda_l \leq \frac{w_i x_1}{x_i} \cdot \frac{2\tilde{U}^*}{w_1} \leq \frac{2w_N}{x_N} \cdot \frac{x_N}{w_N} \cdot 2U^* = 4\tilde{U}^*. \tag{3.8}$$

Specially, if all w_n's are integer power of 2, we have $\frac{x_i}{x_N} = \frac{w_i}{w_N}$, then the system uncertainty bound is reduced to $2\tilde{U}^*$ in both cases. QED.

Step 4: Finally, we prove the bound on the overall system uncertainty, including the uncertainty caused by intervals between detections and the detection latencies.

Proof. We use t_f to denote the full-frame detection latency. We still use $\tilde{U}^*$ to denote the minimum system uncertainty caused by intervals between detections, and use U^* to denote the overall minimum system uncertainty on all feasible schedules. We define U as the overall system uncertainty by BPB schedule, and $\tilde{U}$ as its corresponding uncertainty caused by intervals between detection tasks. We have,

$$U = \max\{U_1, \ldots, U_N\} \leq \max\{\tilde{U}_1 + w_1 t_f, \ldots, \tilde{U}_N + w_N t_f\} \leq \max\{\tilde{U}_1, \ldots, \tilde{U}_N\} + w_1 t_f \tag{3.9}$$

$$\leq 4\tilde{U}^* + w_1 t_f \leq 4U^* + U^* = 5U^*. \tag{3.10}$$

On one hand, $\tilde{U}^*$ is apparently smaller than or equal to system uncertainty caused by intervals between detections in U^*, thus $\tilde{U}^* \leq U^*$; on the other hand, since every schedule includes the full-frame detection, which induces uncertainty $w_1 t_f$ on object $\mathcal{O}_1$, we have $w_1 t_f \leq U^*$. The proof follows. Similarly, when all w_n's are integer power of 2, we have $U \leq 3U^*$. This completes the proof of Theorem 3.1. QED.

3.5 EMPIRICAL OPTIMIZATION

In this section, we list some practical considerations and empirical optimizations we performed in our implementation.

New Object Arrival: We previously did not consider new object arrivals within each horizon. We show in Figure 3.9 that there is no object arrival or departure in most ($\sim 80\%$) frames. Most new objects have very small sizes so only cause minor extra workload. If the detector cannot find one object after W scheduled partial-frame detections, we believe the object has left the view and will stop tracking it. The slots for these objects, together with

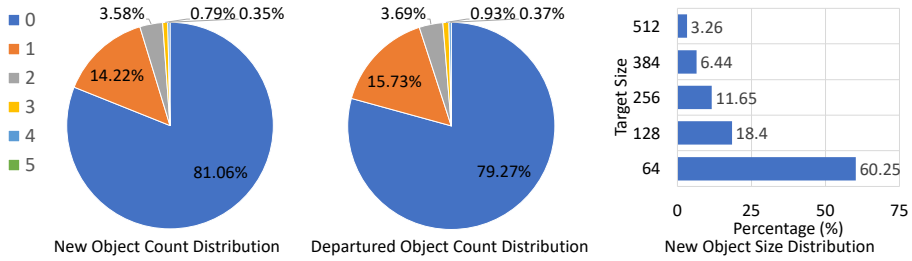


Figure 3.9: Distributions on number of **newly arrived objects** and **departured objects**, as well as the **(quantized) new object size distribution**, at each frame. Results obtained on Waymo Open dataset [31].

Figure 3.10: An example of new pixels in the current frame highlighted in red.

the idle slots in incomplete batches, can be used to schedule the new object regions. To (roughly) localize new objects, we apply a lightweight mechanism based on optical flow. We define the pixels in the new frame that are not mapped to any pixel in the previous frame as the newly appeared pixel, and then use connected component analysis [101] to extract new object regions (*i.e.*, clusters of new pixels). An example is shown in Figure 3.10. Some new objects were also detected by previously scheduled partial-frame detection tasks. Finally, we can set a short scheduling horizon to further reduce the impact of new objects.

Downsizing Large Objects: For the tracked large objects, we can safely downsize their resolutions without affecting the detection quality, because large objects are known to be easy to detect [102, 103]. We set an upper bound (*e.g.*, 384) on the target sizes, where the candidate regions larger than this size will be downsized and fit into the size.

Partial Region Merge: If two quantized candidate regions are highly overlapped, it will be beneficial to merge them into one region, so that we can avoid repetitively scanning the same area. In our implementation, if there is an unscheduled region that the overlap ratio between it and a scheduled region is above a threshold I, we use the merged region to replace the original candidate region. This also helps reduce redundant detections, where different parts of an object are detected as separate objects after slicing.

Bounding Box Filtering: After image slicing, some objects inevitably span across the boundary of several candidate regions. Multiple detections can be produced by the detector on different parts of the same object. We perform a bounding box filtering procedure, as a

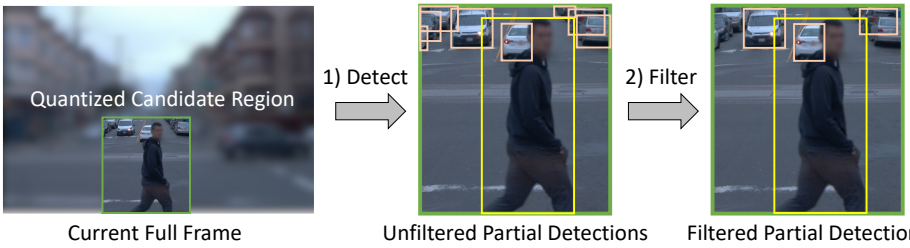


Figure 3.11: An example of bounding box filtering. Note we preserve the yellow box although its bottom edge lies on the bottom border of the partial image, because the partial image bottom coincides with the full image bottom.

postprocessing step, to remove fragmented detections. Specifically, we will remove detected bounding boxes that lie on the partial image boundaries, unless the partial image boundaries coincide with the full image boundaries. We provide an illustrative example in Figure 3.11. Intact redundant detections can be easily removed by the non-maximum suppression (NMS) step of the detector.

3.6 EVALUATION

In this section, we evaluate the effectiveness and efficiency of the proposed architecture on an NVIDIA Jetson Xavier board with a real-world self-driving dataset.

3.6.1 Experimental Setup

Hardware Platform: All experiments are conducted on an NVIDIA Jetson Xavier SoC, which is designed for automotive platforms. It is equipped with an 8-core Carmel Arm v8.2 64-bit CPU, a 512-core Volta GPU, and 32 GB memory. The mode is set as MAXN with the maximum CPU/GPU/memory frequency budget.

Dataset: Our experiment is performed on the Waymo Open Dataset [31], a large-scale autonomous driving dataset collected by Waymo self-driving cars in diverse geographies and conditions. It consists of driving video segments of 20s each, collected by onboard cameras at 10Hz with resolution 1920×1280. Only front camera data is used.

Neural Network for Detection: We use the YOLOv5[7] model in PyTorch as the object detection network, which was pretrained on the general-purpose COCO [32] dataset. It provides a set of model configurations with different depth and width. We specifically use the default large config in the evaluation, with depth and width multipliers both set to 1. The model precision is set at FP16 (*i.e.*, half precision). We profile the YOLO inference

[7] `https://github.com/ultralytics/yolov5`

latency with different target sizes on the Jetson Xavier in advance, and feed them into our scheduling algorithm as input.

Workload Manipulation: Unless otherwise indicated, we choose our scheduling horizon to be 10 frames, and manually change the time interval P between two consecutive frame arrivals to induce different workload. Intuitively, a shorter frame interval leads to a higher scheduling load. Our experiments use three interval lengths (150ms, 100ms, and 70ms) to denote the easy, moderate, and hard scheduling situations (corresponding to frame rates of roughly 6.67Hz, 10Hz, and 14Hz).

Object Criticality: We first assign a static criticality to each object class, and then multiply the class criticality with an approximated distance-based criticality. For class criticality, we manually set the value for each class to simulate how humans prioritize different types of object. For example, class "human" has a much higher criticality than class "vehicle". Within each class, we approximately train a (linear) *distance prediction model* in offline stage. During the training, we extract the object distance (either from LiDAR, or image based distance estimation tools), and regress a linear prediction model between object distance and the bounding box size. A separate distance prediction model is trained for each class. In online inference stage, we use the trained distance prediction model to approximately estimate the object distance based on the detected bounding box size. The two factors together, decide the object criticality. During the evaluation, we separately evaluate the detection performance for overall objects, and critical (i.e., close) objects within each object class.

Evaluation Metrics: Our metrics distinguish between performance of *detection*, *localization*, and *classification*. Here *detection* is interpreted to mean the discovery of whether an object exists (at a location) or not, regardless of type. For example, one might want to detect that an obstacle is in the way, which is a separate challenge from identifying the type of obstacle. *Localization* means identifying the position of the object. Finally, *classification* is the process of identifying object type. Given a list of detections and a list of groundtruth object locations, we match the detections with the groundtruth objects based on their bounding box overlaps. A detection is said to be matched with a groundtruth object if their IoU ratio is larger than a predefined threshold (*i.e.*, $IoU > IoU_{thre}$, set as 0.5 in this paper), in which case we say that the object is *successfully detected*. The following set of metrics are then defined:

- **Detection Recall (DR):** The ratio between the number of successful detections (matched with groundtruth objects) and the count of all groundtruth objects.
- **Detection Precision (DP):** The ratio between the number of successful detections

Table 3.1: YOLOv5 Performance on Waymo Dataset. All values in this table are in percent, except the latency.

Model	Ove. Det. Rec.	Ove. Det. Pre.	Ove. Cls. Acc.	Ove. Loc. Err.
YOLOv5l	76.11	87.54	99.37	4.67
	Cri. Det. Rec.	Cri. Det. Pre.	Cri. Cls. Acc.	Cri. Loc. Err
	97.36	80.77	99.86	2.67
	Ove. mAP	Cri. mAP	Xavier Latency	
	65.19	94.70	239ms	

(matched with groundtruth objects) and the count of all detections.

- **Classification Accuracy (CA):** For each successful detection, we test whether the predicted object class is correct and report ratio of correct classifications.
- **Localization Error (LE):** For each successful detection, its location error is the distance between the estimated and ground truth object center points, as fraction of the object size (*i.e.*, diagonal length).
- **Mean average Precision (mAP):** It is used as an end-to-end metric, which simultaneously captures the error in both detection and classification. An open sourced mAP evaluation engine[8] is used.

We separately evaluate and report the performance on overall objects and critical objects in following evaluation. The overall YOLO performance on Waymo dataset, obtained without time constraint, is listed in Table 3.1, which naturally serves the ceiling condition for the proposed framework.

3.6.2 Impact of Image Downsizing

One natural question is, *why not just downsize the images such that full-frame detections can run in real time*. In this section, we investigate the impact of image downsizing on achieved detection quality and inference latency. We downsize the images to different resolutions, and evaluate the mean average precision (mAP) on objects of different sizes, as well as the corresponding latency. The results are shown in Figure 3.12. We have the following observations: First, reducing image resolutions generally results in degraded detection quality, especially on small objects. However, small objects can be prevalent and critical in driving scenarios. Second, large objects are more robust to image downsizing, so after image slicing, we can safely reduce the resolution of candidate regions containing large objects to achieve better efficiency.

[8] `https://github.com/Cartucho/mAP`

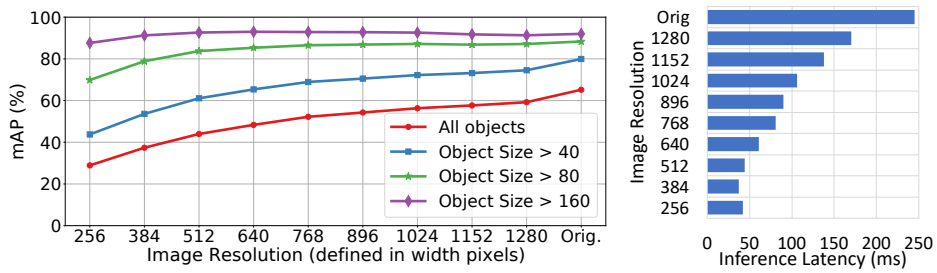
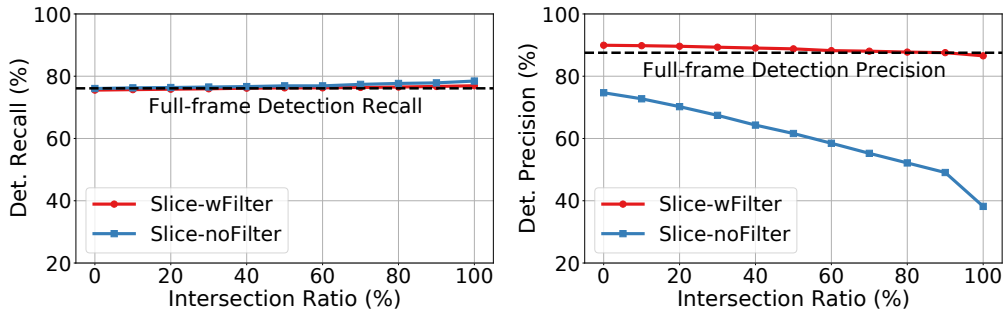


Figure 3.12: Impact of image resolution on detection quality and inference latency.



(a) Detection recall.

(b) Detection precision.

Figure 3.13: Impact of slicing and bounding box filtering.

3.6.3 Impact of Slicing

Next, we evaluate the impact of the tracking-based slicing module on the achieved detection quality. A good slicing module should be lossless and lead to no degradation in detection quality. To isolate the impact of slicing from scheduling algorithms, we run detections on all sliced candidate regions. Besides, two practical optimizations affect the detection quality: (i) bounding box filtering, and (ii) candidate region merges. We evaluate the detection recall and precision with/out bounding box filtering, under different candidate region merge criteria (*i.e.*, the intersection ratios) in Figure 3.13. First, we found that image slicing mainly affects the detection precision, because more false positive detections (corresponding to fragmented object parts) are generated after slicing. The region merge does help partially improve detection precision, especially when more merges (*i.e.*, smaller intersection ratio) are applied. However, bounding box filtering is the key factor that makes the slicing lossless. The red curve of Figure 3.13(b) indicates the slicing module shows almost no degradation on detection precision under different merging criteria after applying bounding box filtering. The fragmented detections are effectively removed. Its detection precision is even slightly higher than full-frame detections when more merges are applied, because slicing helps reduce false positive detections caused by YOLO that span across partial frame boundaries. Second, the detection recall is not affected, because the partial frames completely cover the groundtruth objects.

Table 3.2: Detection and classification quality with different **tracking algorithms**. h denotes the height of the object. All values in this table are reported in percentage.

Frame Interval	Scheduler	Regular Objects					Critical Objects				
		NDR	NDP	CA	LE	NmAP	NDR	NDP	CA	LE	NmAP
$P = 150ms$	Optical Flow	98.07	99.79	99.49	4.89	96.07	96.38	103.89	100.00	2.55	94.95
	Kalman Filter	94.92	99.81	99.47	5.01	94.01	89.13	102.54	99.95	2.79	90.38
	OF - KF	**3.15**	**-0.02**	**0.02**	**-0.12**	**2.06**	**7.25**	**1.35**	**0.05**	**-0.24**	**4.57**
$P = 100ms$	Optical Flow	94.90	97.25	99.46	5.28	89.33	94.58	102.36	99.95	2.94	90.65
	Kalman Filter	91.22	96.98	99.46	5.51	88.13	88.16	100.49	99.95	3.09	87.13
	OF - KF	**3.68**	**0.27**	**0**	**-0.23**	**1.20**	**6.42**	**1.87**	**0**	**-0.15**	**3.52**
$P = 70ms$	Optical Flow	90.93	96.14	99.48	5.61	82.91	90.60	101.71	100.00	2.92	89.80
	Kalman Filter	84.70	93.21	99.35	6.06	79.60	85.46	98.49	99.89	3.36	84.75
	OF - KF	**6.13**	**2.93**	**0.13**	**-0.45**	**3.31**	**5.14**	**3.22**	**0.11**	**-0.44**	**5.05**

3.6.4 Tracking Algorithm

We next validate the choice of tracking algorithm. We compare the optical flow tracker with a state-of-the-art tracking algorithm, SORT [33], which essentially uses a Kalman filter to estimate object motions between frames. It extrapolates future object motions from the past trajectories, thus is not sensitive to unexpected object motions. The results are presented in Table 3.2. Normalized results are reported for detection recall/precision and mAP. We separately show the results on overall objects and critical objects, under three different workloads. We found that optical flow works better than Kalman filter in all metrics under each workload. The gap between their detection recall on overall objects increases as we reduce the frame interval, because we need to rely more on the tracking algorithm to predict object locations when there is no available GPU resource to run their partial frame detection tasks. The localization error also correspondingly increases. We further inspect the detection quality on critical (large) objects. Larger objects are typically easier to detect but harder to track, because the relative movement between the objects and the observer can lead to significant changes on their visual appearance, which further poses more challenge to the tracking algorithm. Thus, the choice of tracking algorithm makes a bigger difference on detection recall of critical objects. We conclude that optical flow is the better candidate for the tracking-based frame slicer.

3.6.5 Scheduling Algorithm Comparison

Baselines: We compare the following scheduling algorithms in this experiment.

- **Frame Drop (FD):** It always runs full detection on the most recent frame, and skips the remaining frames.

Table 3.3: Detection and classification quality with different **scheduling algorithms**. h denotes the height of the object. All values in this table are reported in percentage.

Frame Interval	Scheduler	Regular Objects					Critical Objects				
		NDR	NDP	CA	LE	NmAP	NDR	NDP	CA	LE	NmAP
$P = 150ms$	FD	61.2	103.52	99.43	4.68	61.38	61.10	104.91	99.92	2.68	61.13
	GR-NB	89.00	97.79	99.44	5.43	87.69	93.44	102.06	100.00	2.42	91.03
	GR-B	97.32	100.00	99.43	5.00	96.06	95.28	103.57	100.00	2.36	94.36
	PB	**98.07**	**99.79**	**99.49**	**4.89**	**96.07**	**96.38**	**103.89**	**100.00**	**2.55**	**94.95**
$P = 100ms$	FD	40.99	103.94	99.42	4.67	41.22	40.59	105.19	99.89	2.80	40.72
	GR-NB	82.09	94.91	99.44	5.94	78.99	87.76	99.68	100.00	2.91	87.65
	GR-B	88.86	97.24	99.43	5.56	85.77	90.88	100.89	100.00	2.60	90.57
	PB	**94.90**	**97.25**	**99.46**	**5.28**	**89.33**	**94.59**	**102.36**	**99.95**	**2.94**	**90.65**
$P = 70ms$	FD	28.63	103.68	99.43	4.68	28.79	28.57	80.77	99.68	2.73	28.68
	GR-NB	74.58	91.10	99.46	6.65	64.32	72.62	90.86	99.87	4.36	71.52
	GR-B	76.02	91.34	99.41	6.57	67.10	73.40	91.49	99.88	4.32	73.92
	PB	**90.93**	**96.20**	**99.48**	**5.61**	**82.48**	**91.01**	**96.19**	**100.00**	**2.92**	**89.81**

- **Greedy without batching (GR-NB):** At each frame, it always schedules the partial frame detection task with the highest weighted uncertainty. No batching is performed. Note the same candidate region merge strategy is used in GR-NB, GR-B and PB algorithms.
- **Greedy with batching (GR-B) [38]:** It always schedules the partial frame detection task with the highest weighted uncertainty, and batches as many partial frame detection tasks as possible with the same size.
- **Proportional balancing policy (PB):** The proposed scheduling algorithm that controls the detection frequency of each object according to their criticality and uncertainty growth, with exploiting task batching.

Results: The corresponding results are summarized in Table 3.3. Similarly, normalized results are reported for detection recall/precision and mAP. We test the scheduling algorithms at different workloads (*i.e.*, frame intervals). We have the following observations: First, slicing-and-detection paradigm is always better thanks skipping frames indicated by the fact that it achieves much higher detection recall, no matter which scheduling algorithm is used. On the contrary, FD missed a large amount of objects. FD shows high and roughly constant detection precision under different workload because it does not output any predicted object locations from the tracker. Second, among the compared scheduling algorithms, our PB policy shows better detection quality than the two greedy baselines on both overall objects and critical objects, especially when the task load is high, *i.e.*, $P = 70ms$. PB effectively maintains a high detection recall when we reduce the frame interval, while

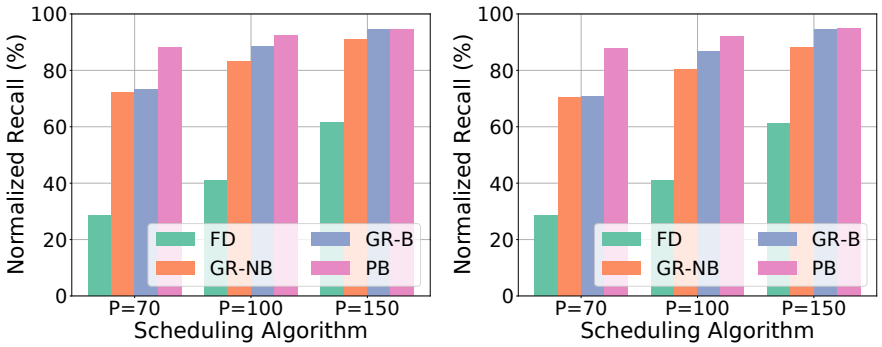
the greedy baselines can not. This is especially important because it shows our PB policy misses much less objects than the baselines, when the situation is critical. The improvement comes from the fact that PB calculates the task schedule for the entire horizon in advance, thus achieving better resource and time utilization. On the contrary, GR-B and GR-NB execute in an ad-hoc manner at each frame without intellectual planning performed on partial frame detection tasks. Third, task batching does not play a significant role in improving the detection quality, which is different from the result in [38]. GR-B does not show a huge advantage over GR-NB in most cases. This is because we consider the freshness of information contained in image frames, and do not buffer partial frame detection tasks from the stale frames. Instead, once the new frame arrives, we directly search the objects at the new frame, using the up-to-date expanded candidate regions. Fourth, once the detector finds a presented object, the classification problem is relatively easy. As we can see in Table 3.3, almost no misclassification happens.

3.6.6 Correspondence to System Safety Considerations

In previous experiments, we assumed that large objects are closer and more critical to system safety. In this experiment, we directly test whether the PB policy can provide better response to physically close objects. We use the object distance information extracted from the groundtruth LiDAR range images for the evaluation purpose. Intuitively, to ensure better safety, we do not want to miss detections of any close objects so high recalls on close objects are preferred. We compute the following two recall metrics: one for the closest object at each frame, and the other for all objects within 20m at each frame. The corresponding results are presented in Figure 3.14. We show the normalized detection recall, which is the ratio between the achieved detection recall and the detection recall of full frame detections. The results indicate that the PB policy is most effective in responding to close objects under high workload because it achieves significantly higher recall than the baselines when $P = 70ms$. Under low workload (*i.e.*, $P = 150ms$), GR-B and PB show similarly near-optimal detection recalls on close objects.

3.6.7 Breakdown of Overhead Quantification

We next breakdown the latency overhead induced by each step in our processing pipeline. The results are given in Table 3.4. The PB policy runs pretty fast and only executes once per scheduling horizon. Since optical flow estimation does not depend on any other processing steps, it runs in a separate process on the CPU and poses no overhead to the detection



(a) The closest object. (b) Close objects within <20m.

Figure 3.14: Responsiveness to physically close objects.

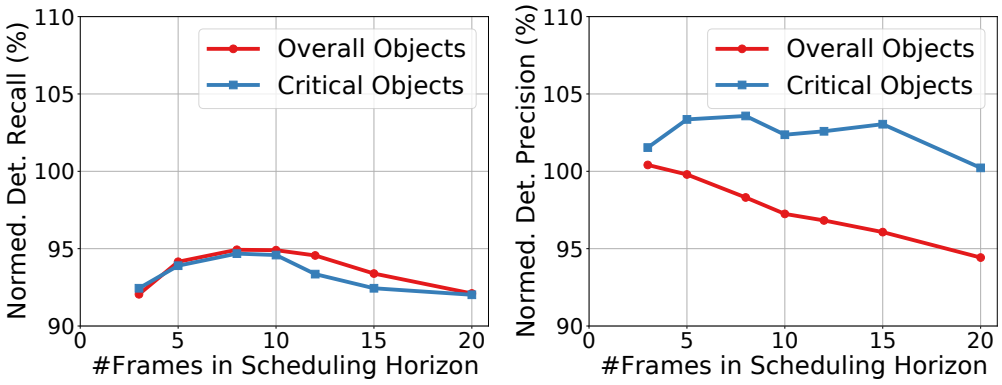
Table 3.4: Breakdown Latency Overhead

PB Policy	Flow Est.	Tracker Prediction	Runtime Optimization	Postprocessing
6.47ms	12.21ms	9.22ms	8.28ms	8.08ms

pipeline on GPU. The tracker prediction step uses the estimated flow map to predict the new object locations, and the expanded candidate regions. The runtime optimization step performs the empirical optimizations on top of the initial schedule generated by the PB policy (*e.g.*, region merge, region replacement, and batch formulation). Finally, the postprocessing step filters the generated detections and maps the remaining detections to the full frame locations.

3.6.8 Choice of Scheduling Horizon Length

Finally, we investigate the impact of the scheduling horizon length on achieved detection quality. We vary the number of frames in a scheduling horizon from 3 to 20, and see how the detection recall and detection precision correspondingly change. We set the frame arrival interval $P = 100ms$. The results are given in Figure 4.9. The proposed PB policy is generally resilient to the length of scheduling horizons and does not show a large variation on detection recall under different settings. We also notice that moderate lengths show relatively better detection recall on both overall objects and critical objects. When the scheduling horizon is too short, the full frame detection frequently steps in and does not leave enough time to run necessary partial frame detections. Critical objects may escape from the tracking window. When the scheduling horizon is too long, we do not have timely updates on the object presence and criticality, and may waste time tracking objects that are not critical anymore. In the detection precision figure, we found the detection precision decreases monotonically



(a) Detection Recall.

(b) Detection Precision.

Figure 3.15: The impact of the choice of #frames in a scheduling horizon.

as the horizon length increases. This also reflects the staleness of scheduling decisions when the scheduling horizon is too long. Therefore, shorter horizons ensures better timeliness on scheduling decisions, while longer horizons provide more flexibility and space for scheduling. We believe choosing a moderate (10 frames or 1 second) is most beneficial to achieving a good tradeoff between scheduling flexibility and decision timeliness.

3.7 RELATED WORK

In this section, we briefly review recent literature on real-time machine inference and exploitation of temporal correlations in video object detection.

3.7.1 Real-time Machine Inference

Most previous research to support real-time machine inference focused on compressing neural network models to reduce the inference latency [48, 49, 51, 54, 104]. DeepIoT [51] compressed neural network structures into smaller dense matrices by only keeping non-redundant hidden elements. A follow-up work, FastDeepIoT [48], further exploited run-time non-linearities in the neural network configuration space to accelerate network inference. Zhou *et al.* [49] investigated the relationship between model performance and parameter quantization. In the approaches exemplified above, the compression policy is typically fixed and cannot provide the flexibility to dynamically adjust computation at run-time to meet deadlines. At meanwhile, with the advance of cloud computing, cloud offloading [96] has been proposed as a new paradigm for speeding up deep neural network inference. Since we target at the autonomous driving applications, a fast and reliable networking connection

is unavailable in many driving scenarios. Recently, real-time scheduling has emerged as a key challenge in AI-empowered perception systems. Real-time AI scheduling is generally divided into three categories: (i) system-level scheduling; (ii) neural network-level scheduling; and (iii) data-level scheduling. System-level scheduling algorithms try to optimize CPU-GPU interactions by appropriately allocating and pipelining the computational stages to reduce the end-to-end processing latency [56, 57, 58, 105]. In contrast, neural network-level scheduling algorithms dynamically adjust the utilized neural structures to meet inference deadlines [83]. To implement the "any time" neural networks, Bateni *et al.* [83] chose among a set of approximation techniques at each network layer to achieve the minimal accuracy degradation within the deadline. Several recent works focused on dynamically refactoring the wiring connections between network layers so that one can skip some intermediate layers or early exit the inference with possibly imprecise results [85, 86, 106]. Finally, the data-level scheduling algorithms first slice the data into partial regions, then schedule the processing of such partial regions according to their criticality. Recent approaches [38, 107] rely on an external ranging-sensor (*e.g.*, LiDAR) to determine regions of interest and assess criticality. One drawback of existing approaches [38, 107] lies in their reliance on the external ranging sensor, which may be not be an option in some autonomous systems (*e.g.*, Tesla has famously opposed using LiDAR[9]), and also requires high quality calibration and synchronization between the sensors to ensure reliable output. Instead, in this paper, we describe a self-cueing system that only relies on past detections and a tracking module to determine regions of interest that need to be processed in the upcoming image frame.

3.7.2 Temporal Correlations in Video Object Detection

Video temporal correlations have been extensively studied to improve the efficiency of continuous object detection. Some papers, including [108, 109, 110, 111], rely on motion vectors between consecutive frames to reduce the network depth to extract features on new frames. They utilized motion vectors to map (part of) past features into the new frame. Buckler *et al.* [112] proposed an optical flow-based hardware solution to propagate latent features from previous frames to the new frame. However, the uncertainty in the estimated motions were not counted. Some works also leverage pairwise image differences to guide an object detector to focus on changing areas in the new frame [113, 114]. However, they are only applicable to statically mounted cameras, which no longer holds in autonomous driving systems. Song *et al.* [115] applied different quantization levels to process regions with different sensitivity on the same frame, which was only limited to image classification models.

[9]For example, see https://provscons.com/why-doesnt-tesla-use-lidar/

Both Kumar *et al.* [116] and Mao *et al.* [117] proposed to use object tracker projections to extract regions of interest in the new frame. However, neither paid attention to the scheduling of processing tasks on extracted regions. To the best of our knowledge, we are the first to integrate tracking-based image slicing and corresponding task scheduling to optimize the efficiency of visual perception.

3.8 LIMITATIONS AND DISCUSSION

There are also associated limitations in the self-cueing scheduling framework. First, when based on solely the image information, the correspondence of the prioritization mechanism to the physical world is not as strong as the external-cueing mechanisms. The main objective is set to optimizing the DNN model accuracy, which does not has a direct association with autonomous driving safety. To improve it, either LiDAR-based distance or vision-based distance estimation approaches should be integrated. Second, the scheduled full-frame detection becomes the efficiency bottleneck, which can affect the following frame processing in some cases. However, it is still irreplaceable in achieve comprehensive environment perception with high fidelity. Therefore, the key question becomes how to better reduce the negative effect caused by the long execution time of the full-frame detection.

CHAPTER 4: ATTENTION-BASED MULTI-CAMERA SCHEDULING

4.1 OVERVIEW

This work introduces a real-time multi-view camera scheduling framework for AI-enabled live video analytics that optimizes frame processing speed at the edge. Processing live videos at the edge provides better latency, scalability, and privacy. In an edge processing framework, most of the videos collected can be immediately discarded without consuming further resources, except when events are detected that require further analysis. Applications include security surveillance, accident detection, traffic-violation recording, and elderly monitoring. While advances in modern deep learning techniques greatly improve the quality of video analytics, they also impose a high computational cost. We consider applications that require a real-time response such as catching a traffic violation in action. In these applications, applying deep learning techniques to live video analytics poses efficiency challenges that our system aims to resolve.

Specifically, we consider scenarios where a set of smart cameras are deployed around a local area, such as a mall or a traffic intersection. The cameras perceive the environment with complementary but partially overlapped views so one object may be observed by multiple cameras simultaneously. Each camera has access to limited onboard computing capacity, most notably including a lower-end GPU. We consider *neural network-based object detection and tracking* as the workload to be scheduled on such GPUs. We follow a standard tracking-by-detection paradigm where tracking relies on repetitive calls to object detection modules, such as YOLO [87], to update the exact locations of tracked objects.

Significant efforts were spent on optimizing the efficiency of video analytics. Most of them focus on centralized video processing [118, 119, 120, 121], where camera video streams are uploaded to a powerful cloud server. Network bandwidth is considered as the major efficiency bottleneck. Even with fast and stable network connections, such systems still suffer from longer response latency than those where the need for video offloading is removed. Some recent work [122, 123] directly optimizes the processing efficiency on cameras by exploring cross-camera correlations. However, they improve the efficiency by turning off cameras that are unrelated to a specific query, which does not speed up the execution on cameras that remain on.

In this work, we aim at achieving neural-network-based real-time live video processing directly at edge cameras with limited computational capacity and network bandwidth, by exploiting the spatial-temporal correlations in multi-view video streams. Based on the ob-

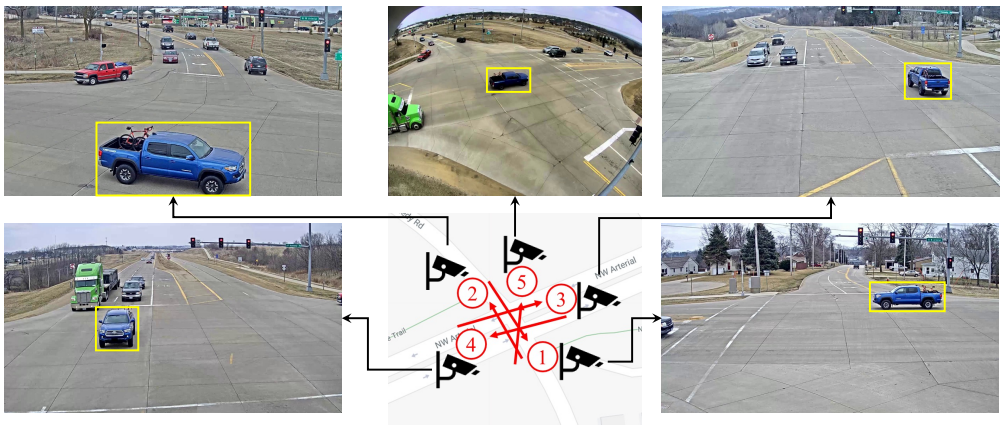


Figure 4.1: An example showing the camera view overlaps. We use yellow boxes to mark an object observable to every camera.

servation that the location of the same object does not change significantly over consecutive video frames, we run object detection on full camera frames at a significantly *lower frequency* than the frame rate (*e.g.*, only once per second), and inspect a set of object-based *partial frames* between full frame detections. Partial frames are sliced around estimated object locations to help locate the objects of interest, while avoiding repeated scanning of more static regions (*e.g.*, the sky and background). Since multiple partial frames can be extracted from a full frame, we apply the task batching in [38], where multiple input images with the same size can be submitted as a single batch to the GPU, such that the batch latency is much lower than processing the images sequentially. When an object is visible from multiple views, we only need one camera to keep track of its location, raising the problem of allocating object tracking tasks to cameras to optimize the neural network processing speed. This problem becomes more challenging when the tasks with the same size can be batched. We formulate it as a multi-view scheduling (MVS) problem. We prove it to be strongly NP-hard, and propose a batch-aware load-balanced scheduling (BALB) algorithm to approximately solve it.

Our multi-view scheduling algorithm operates on two stages. First, all cameras communicate with a *central scheduler*.[1] Specifically, after a full frame detection, each camera uploads its list of detected objects to the central scheduler. The central scheduler first associates objects across cameras, and then runs a *central stage* of the BALB algorithm to derive an initial assignment between objects and cameras. Both the heterogeneous neural network processing speed at cameras and task batching opportunities are considered in the optimization. Second, in order to deal with newly appeared and disappeared objects, in between

[1]We do not pose assumption on the computing capacity of the central scheduler since no actual detection happens there. We can choose a camera to work as the central scheduler.

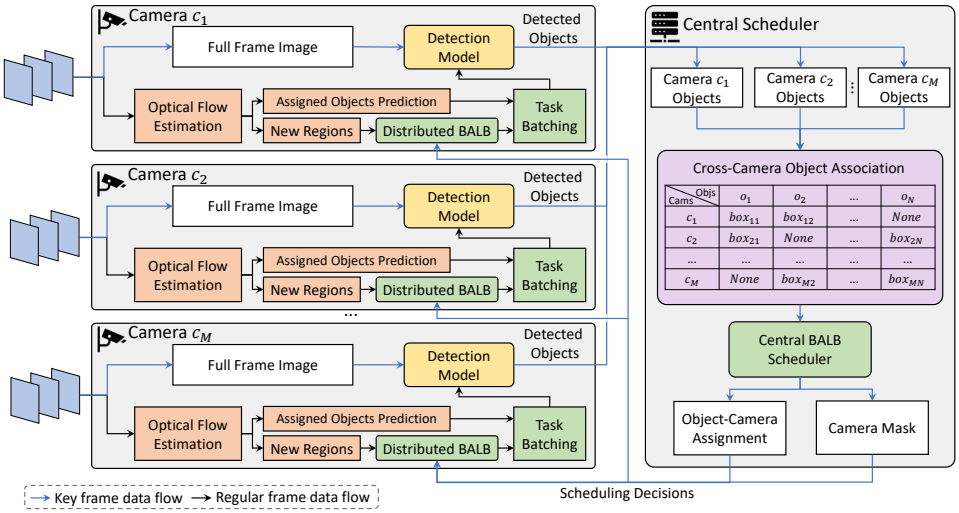


Figure 4.2: Overview of the proposed framework. For the block colors: The tracking-based slicing module is highlighted in orange, the cross-camera object association module is shown in purple, and the BALB scheduler related operations are highlighted in green.

full-frame detections, each camera independently runs a *distributed stage* of the BALB algorithm to decide whether to track a new object or not. Although the distributed decisions may cause load imbalance among cameras, it reduces the overhead of centralized scheduling and the imbalance is quickly corrected by the next central stage in a few frames.

We implement the proposed scheduling framework on a testbed consisting of multiple NVIDIA Jetson boards, including Jetson Nano, Jetson TX2 and Jetson Xavier. We comprehensively evaluate the system performance with AI City Challenge 2021 (AIC21) dataset [124, 125]. The results show that our system consistently improves the frame processing speed by by 2.35× to 4.97×, at the cost of minor degradation on detection quality. When compared to a video offloading system, it achieves a similar processing throughput but with a much lower response latency and network bandwidth cost.

4.2 SYSTEM ARCHITECTURE

In this section, we first give an overview of the proposed scheduling framework, then briefly introduce each component constituting the system.

4.2.1 System Overview

Assume we have a set of static cameras deployed around a local area, such as a shopping mall or a traffic intersection. The cameras share complementary views to obtain compre-

hensive perception of the target environment. Further, their views are partially overlapped so one object may be simultaneously captured by multiple cameras. One such example is shown in Figure 4.1. The cameras run object detection and tracking locally. Each object needs to be localized, classified, and tracked at every frame. Since deep neural detection models, like YOLO [87], are resource consuming, we regard them as the main workload to optimize. Each camera has limited computing capacity, such as a low-end GPU. Computing resources may be heterogeneous, so the neural network processing speed may vary among cameras. The detection model accepts input images with various sizes, where smaller images achieve shorter execution times.

We assume that, on one hand, the onboard GPUs are not powerful enough to process every full camera frame in real-time, because running the detection model takes longer than the inter-frame interval. On the other hand, the cameras do not have enough network bandwidth to stream high-resolution videos to the cloud in real-time for centralized processing.

We optimize the neural network processing speed at cameras by taking advantage of the spatial-temporal correlations in multi-view video streams, and by applying effective task batchings on GPUs. Instead of running full frame detection on every sampled image, we only run it on some frames (which we call the *key frames*) at a fixed low frequency (*e.g.*, once per second). At the remaining frames, which we call the *regular frames*, we insert a set of object-oriented partial frame detections to search the objects around their predicted locations. When an object appears in multiple cameras, we only schedule one camera to track it. The object-camera assignment considers both the load balancing among cameras and GPU task batching opportunities for the input images with the same size. An overview of the proposed system is given in Figure 6.1. It is comprised of the following three main modules:

- **Tracking-based slicing:** We slice frames into partial regions around predicted object locations, computed by the tracking algorithm, thus reducing neural network input.
- **Cross-camera object association:** We associate individually detected objects across cameras to reduce redundant tracking of the same object.
- **Multi-view scheduling:** We decide the subset of objects to track by each camera and their corresponding batching, to minimize the neural network processing time.

4.2.2 Optical Flow-based Tracking and Image Slicing

We follow the general tracking-by-detection paradigm [126], where object tracking is achieved by first detecting all objects in the new frame and then associating them with

previously tracked object trajectories, where applicable. Concrete tracking algorithms in this category differ in how they associate the new detections with the tracked trajectories. We follow the association design of SORT [33] in this work, which purely relies on the overlaps between the new detections and the predicted object locations to perform the association. However, the original Kalman filter-based motion model no longer works in our scenario, because they rely on long observation history to be reliable. In our case, we immediately rely on the predicted object locations to slice the partial frames even when they have been only detected once in the latest full frame detection. Improper slicing may make the object detection model fail to find the complete bounding box of the object, which will result in further cascaded error in the detection and slicing iterations until we miss the object completely. Thus, we replace Kalman filter with an optical flow model [97], which automatically estimates the pixel motions between two input images. It is a memoryless model that does not rely on any history information except the object location in the previous frame. Motion vector at each single pixel is generated, and the object motion is estimated as the median value among pixel motions contained in its previous bounding box. This is based on the assumption that the object pixels dominate the bounding box compared to the static background pixels. Since we can only batch input images with the same size, we expand the predicted object locations to the nearest size in a quantized set to increase the batching opportunities. The quantized size is fixed for each object within a scheduling horizon, and downsizing is performed if the object size grows beyond it.

In addition, the estimated pixel motions can be used to roughly detect newly appeared objects at regular frames. Since cameras are static, the relative movement between the object and the cameras is completely decided by the object movements. We define the clusters of moving pixels that do not belong to any existing objects as a *new region*, where a new object may appear. We also feed these regions into the object detection model to localize new objects. By doing so, we can detect new objects at their first appearance, instead of waiting until the next full frame detection.

4.2.3 Cross-Camera Object Association

We next explain how we perform cross-camera object associations. Specifically, the algorithm maps a detected object by one camera to the detected objects by the remaining cameras. We are aware that the cross-camera data association and tracking is an active research area [127]. This topic is outside the scope of this thesis. In this Chapter, we assume that the association problem has been solved by others, and we will focus on the multi-view real-time scheduling part.

4.2.4 Multi-View Scheduling

After we obtain the associated object list, the remaining problem becomes how to assign objects to cameras for tracking, such that the maximum execution time among cameras is minimized. We formulate it as a multi-view scheduling (MVS) problem, and correspondingly propose a two-stage batch-aware load-balanced (BALB) scheduling algorithm to solve it. It considers both the load balancing among cameras and the task batching opportunities on GPU. The cameras track and batch the assigned objects according to the scheduling decisions. The details will be explained in the next section.

4.3 MULTI-VIEW SCHEDULING

In this section, we first describe the task execution model, and then formulate the *multi-view scheduling (MVS) problem*. Finally, we introduce the *batch-aware load-balanced (BALB) algorithm*, to solve the scheduling problem.

4.3.1 Object Detection Task Model

Assume we have a set of M cameras $\mathcal{C} = \{c_1, c_2, \ldots, c_i, \ldots c_M\}$ equipped with heterogeneous computing capacities. They monitor a local area with overlapped views, and run object detection models (*e.g.*, YOLO [87]) (along with a lightweight tracking algorithm) to track the appeared objects. We define the time period between two full frame detections as a *scheduling horizon*, during which a fixed number of $T-1$ frames are captured. At start of the scheduling horizon, a set of N objects $\mathcal{O} = \{o_1, o_2, \ldots, o_j, \ldots, o_N\}$ are identified from the latest full frame detection, and their locations on each camera are obtained. This is achieved by connecting all cameras to a *central scheduler*, and first running the cross-camera object association algorithm to produce the object mappings among cameras. However, the object set $\mathcal{O}$ may evolve during a scheduling horizon since new objects may arrive and existing objects may leave. We want to track the locations of all appeared objects within the scheduling horizon, through scheduling *partial frame detection* tasks on the cameras. We define the *coverage set* $\mathcal{C}_j \subseteq \mathcal{C}$ of object o_j as the subset of cameras that can see it. It can be tracked from any $c \in \mathcal{C}_j$.

Each object o_j is associated with a target size s_{ij} at each camera $c_i \in \mathcal{C}_j$, which defines the size of the partial regions where we will search the object. The target size for the same object can be different among cameras, but is fixed in a scheduling horizon (with possible downsizing). Since task batching on GPUs only accepts input images with the same size.

We quantize the target sizes (by expanding the region) to a limited set $\mathcal{S} = \{s_1, ..., s_K\}$ to increase batching opportunities. Given a target size s, at most B_i^s partial regions can be batched and processed in parallel on camera c_i, which is called the *batch limit* of target size s on c_i. The incurred latency is t_i^s when $b \leq B_i^s$ tasks are batched[2]. Benefit from the parallel computing capability on GPUs, batching significantly increases the neural network processing throughput.

4.3.2 Scheduling Problem Formulation

We assume partial regions of different frames within a scheduling horizon are processed sequentially without considering cross-frame batching. Here we focus on formulating the scheduling problem for a single frame. Before that, we first define the camera load below.

Definition 4.1 (Camera Load)**.** Given a camera c_i, its load L_i is defined as the sum of execution latencies of all its batches belonging to one frame. No preemption is allowed during batch executions.

We further define the *system load* L as the maximum load among all cameras in set $\mathcal{C}$, *i.e.*, $L = \max_i L_i$. The scheduling problem we study is to derive a *feasible assignment* $\mathbf{X}$ between cameras and objects such that the system load is minimized. The feasible assignment is formally defined below.

Definition 4.2 (Feasible Assignment)**.** An assignment between a set of cameras $\mathcal{C}$ and a set of objects $\mathcal{O}$ is a matrix $X \in M \times N$, where $x_{ij} \in \{0, 1\}$ indicates whether camera c_i tracks object o_j. A feasible assignment satisfies the following conditions: (1) Each object is tracked by at least one camera that can see it, *i.e.*, $\sum_{c_i \in \mathcal{C}_j} x_{ij} \geq 1,\ \forall j$. (2) No object can be tracked by a camera that can not see it, *i.e.*, $\sum_{c_i \in \mathcal{C} \setminus \mathcal{C}_j} x_{ij} = 0, \forall j$.

Given a feasible assignment, it would be trivial to convert it to optimal batch sequences at cameras that achieve the minimum load. The camera load only depends on the number of batches for each target size. We derive the optimal batch sequence on each camera by batching objects with the same target size in a greedy manner, which apparently minimizes the number of used batches. Each target size is independently batched. Thus, a feasible assignment uniquely decides the corresponding optimal system load. We then formally define the scheduling problem below.

[2]Although batching too many images would lead to a non-ignorable increase in execution latency, we operate in a region where the execution time changes only slightly with batching (before an inflection point is reached where the slope increases). We correspondingly set the execution time at the batch limit as the batch execution latency t_i^s.

Definition 4.3 (Multi-View Scheduling Problem)**.** The Multi-View Scheduling (MVS) problem asks for a feasible assignment between camera set $\mathcal{C}$ and object set $\mathcal{O}$ such that the system load L is minimized.

We establish the computational complexity of the MVS problem in Claim 4.1.

Claim 4.1. The MVS problem is strongly NP-hard.

Proof. We prove by reducing the bin packing problem to the MVS problem. We first constraint the MVS problem to an identical machine scheduling (IMS) problem by adding the following constraints: 1) The batch limit is always one (*i.e.*, no batching is allowed); 2) Every object can be seen from all cameras, thus can be assigned to any camera; 3) All cameras have the identical processing speed; 4) Each object has the same target size across all cameras, so its execution latency is the same on different cameras. Minimizing the system load in the IMS problem can be converted to an equivalent decision problem: Given a time budget T, can we finish processing assigned objects on all M cameras? If we consider M identical cameras as M bins with capacity T and regard the N objects as N items with their execution latency defined as the item size, then the decision version of assigning objects to cameras becomes a standard bin packing problem, which has been proved to be strongly NP-hard [128]. The claim follows. QED.

We next propose an efficient algorithm that approximately solves the multi-view scheduling problem.

4.3.3 Batch-Aware Load-Balanced Scheduling Algorithm

The fact that the object set $\mathcal{O}$ evolves within a scheduling horizon makes the scheduling even more challenging. A camera is unaware of object changes at other cameras unless the object is also observable to itself. From the optimization perspective, the updated object list at cameras should be uploaded to the central scheduler at every frame, to produce the updated assignment based on the aggregated information. However, the overhead caused by camera-scheduler communication may exceed the benefit of the produced assignment. From the scheduling efficiency perspective, we should design a fully distributed mechanism which runs independently at each camera. However, it completely ignores the load and GPU task batching at each camera, so will produce inferior assignment.

We observe that objects appear and disappear at a low frequency compared to the camera sampling frequency. We thus propose the **Batch-Aware Load-Balanced (BALB)**

Algorithm 4.1: Central Stage BALB Algorithm

Input: Object coverage sets $\mathcal{C}_1, \ldots, \mathcal{C}_N$, camera execution latencies t_i^s, and batch limits B_i^s, $\forall s \in \mathcal{S}, \forall c_i \in \mathcal{C}$.

Output: Feasible assignment $\mathbf{X}$, camera load L

Initialize: $x_{ij} := 0,\ \forall i, j,\ L_i := t_i^{full},\ \forall i$;

Reindex the objects $o_j \in \mathcal{O}$ by non-decreasing order of $|\mathcal{C}_j|$ (ties broken in favor of larger target size);

for $o_j \in \mathcal{O}$ *(after object reindexing)* **do**

  $\mathcal{C}'_j := \{c_i | c_i \in \mathcal{C}_j$ **and** $\exists$ incomplete s_{ij} batch$\}$;

  if $|\mathcal{C}'_j| > 0$ **then**

    $c_{i^*} :=$ The camera in $\mathcal{C}'_j$ with the largest relative capacity in the incomplete s_{i^*j} batch;

    $x_{i^*j} := 1$;

  end

  else

    $c_{i^*} := \arg\min_{c_i \in \mathcal{C}_j} L_i + t_i^{s_{ij}}$;

    $x_{i^*j} := 1$, $L_{i^*} := L_{i^*} + t_{i^*}^{s_{i^*j}}$;

  end

end

Return the assignment $\mathbf{X}$ and the camera load L;

scheduling algorithm that works in two stages. It runs a *central stage* on the central scheduler, and a *distributed stage* on cameras. The central stage is only called once at the start of the scheduling horizon, to produce an initial assignment based on the associated object list. Then, at each regular frame, the cameras independently call the distributed stage to update the assignment if there are changes on the detected objects. The potential imbalance produced by the distributed stage will be corrected by next central stage in a few frames.

Central Stage. After a full frame detection, the cameras upload the list of detected objects to the central scheduler. The central stage algorithm takes the associated object list as input to produce the initial assignment. It works in a batch-aware load-balanced manner to fully exploit the camera view overlaps and task batching mechanisms on GPUs.

We initialize the camera load as their corresponding full frame detection time t_i^{full}, and then try to minimize the maximum camera load through maintaining a good load balancing property among cameras during the process. The idea is motivated by the following observation: For an object o_j, the more cameras can see it, the more flexibility we have in scheduling its tracking. Therefore, we start with assigning objects that are observable from only one camera, which have a deterministic assignment. After that, we gradually assign objects with more flexibility (*i.e.*, larger coverage sets). When assigning an object o_j, we try to maximally batch tasks. We will not start a new batch for o_j as long as there exists a

camera $c_i \in \mathcal{C}_j$ that has an incomplete batch for target size s_{ij}. If there are multiple cameras with incomplete batches, we choose the one with the maximum *batch capacity*, which is defined below.

Definition 4.4 (Batch Capacity)**.** Given an incomplete batch with b batched objects, the batch capacity is defined as $BC = B - b > 0$, where B is the corresponding batch limit.

Otherwise, we have to start a new batch for o_j. In this case, we select the camera that has the minimum updated load after including the new batch. It is different from assigning o_j to the camera with the minimum current load because the neural network processing time for the same object may be different among cameras due to different target sizes and neural network processing speed at cameras. The details of the proposed central stage BALB algorithm are summarized in **Algorithm 4.1**. It has a low computation complexity as $\max(O(N \log N), O(MN))$.

Distributed Stage. The initial assignment produced by the central stage does not consider the evolution of object set $\mathcal{O}$ during the scheduling horizon. The distributed stage independently runs at each camera to update the assignment for newly appeared objects and objects disappeared on their originally assigned cameras. The assignment of the remaining objects are unchanged. The objective of the stage is to first guarantee that each appeared object is tracked by at least one camera, and then optimize the efficiency as much as we can.

Algorithm 4.1 can not be directly used because the cameras do not know the exact load and task batching situations on other cameras. In order to make sure the cameras make consistent decisions, we have to rely on fixed policies that work in a self-organized way without direct communications. Specifically, we sort the cameras in increasing order of their assigned load by Algorithm 4.1, and use that order as the fixed camera priority to assign objects. This order is fixed during the scheduling horizon, and will be updated by the next central stage. Each camera only tracks objects at regions that are unobservable from all higher priority cameras. We apply the policy to the following two cases.

First, we define new objects as objects that arrived after the last key frame. New objects enter the view of different cameras asynchronously. Therefore, we dynamically update their assignment at each frame. At each frame, we first use the cross-camera object association algorithm to decide its coverage set, and then choose the camera with the highest priority to track it.

Second, existing objects exit the view of different cameras asynchronously. For each object that appeared in the previous key frame but was not assigned to this camera, we test whether the object has disappeared on its assigned camera but is still observable from this camera. If yes, we calculate its new coverage set, and select the camera with the highest priority to

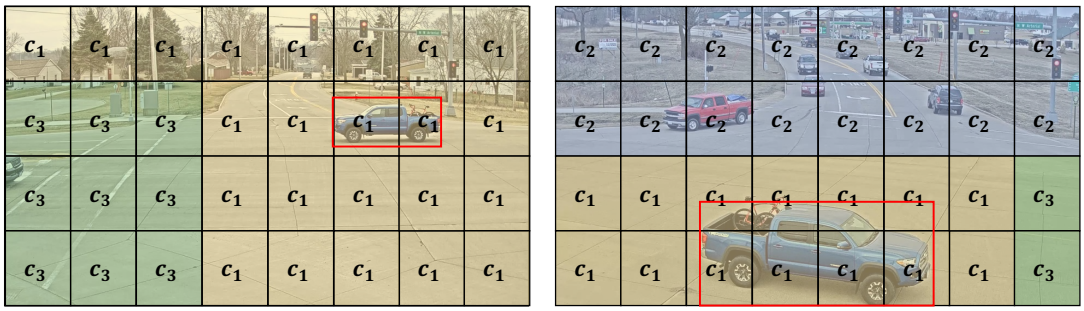


Figure 4.3: Illustration of camera masks. We assume the load-based priority order is: $c_3 > c_1 > c_2$. Each camera only tracks new objects and objects disappeared in assigned cameras at region cells that are unobservable from higher priority cameras. If we regard the blue vehicle as a new object, camera c_1 will track it, and camera c_2 is also aware of the decision.

take over its tracking.

In both cases, the camera will automatically start tracking the object if itself is selected. No communication is performed between cameras. In the implementation, we compute a mask for each camera after the central stage, as shown in Figure 4.3, which indicates regions where the camera should track the aforementioned objects. We first divide the camera frame into a grid of cells, and compute the coverage set for each cell. We then choose the camera with the highest priority to cover the cell. In the distributed stage, the new objects, or objects that exit their assigned cameras, are automatically assigned according to the camera masks.

4.4 EVALUATION

In this section, we evaluate the proposed scheduling framework on a testbed consisting of various Jetson models with the AI City Challenge 2021 (AIC21) dataset.

4.4.1 Implementation and Experimental Setup

Testbed. We implement the system on a heterogeneous edge testbed consisted of 5 NVIDIA Jetson devices: 2× Jetson Xavier, 2× Jetson TX2, and 1× Jetson Nano. Figure 4.4 shows the hardware platform we used. Each device corresponds to a smart camera. They are deployed in an off-campus building, and connected to the central scheduler through wired network (100Mbps Downlink, 20Mbps Uplink). We deploy the central scheduler to a desktop with Intel i9960x CPU located in a campus building. We utilize TCP socket programming for reliable data communication between the edge devices and the central scheduler.

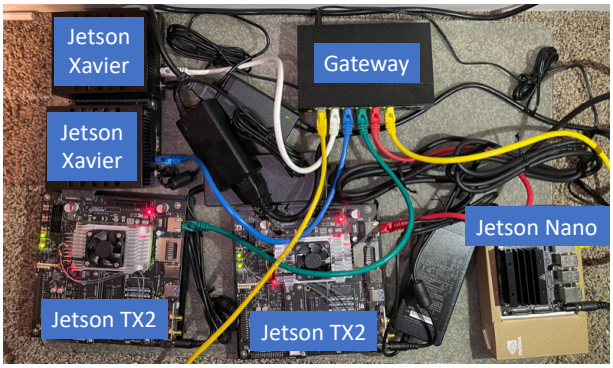


Figure 4.4: Heterogeneous edge testbed.

Table 4.1: Hardware Configuration for Each Scenario

Scenario	Edge Device Configuration
S01	2× Jetson Xaiver, 2× Jetson TX2, 1× Jetson Nano
S02	1× Jetson Xaiver, 1× Jetson Nano
S03	1× Jetson Xaiver, 1× Jetson TX2, 1× Jetson Nano

Dataset. We evaluate the system with AI City Challenge 2021 dataset published by NVIDIA [124, 125]. It consists of traffic camera data collected around traffic intersections and streets. We choose three deployment scenarios to run the experiment, which involve 5, 2, and 3 cameras respectively. The dataset provides synchronized videos from each camera with a frame rate of 10 fps. The full image resolution we use is 1280×704 for regular cameras, and 1280×960 for fisheye cameras. For each camera, we use half length of the video to train the cross-camera object association model with the provided bounding box labels, and use the remaining half for testing. The specific edge device configuration for each evaluation scenario is listed in Table 4.1.

Object Detection Model. We use the YOLOv5[3] model implemented in PyTorch as the object detection network, with pretrained weight on COCO [32] dataset. It is the newest version of YOLO model family and comes with officially released implementation. We choose four sizes for partial frame detection: 64, 128, 256 and 512. Regions larger than 512 are downsampled to 512 as very large objects are easy to be detected. In the offline stage, we profile the YOLO inference time with 200 runs on each Jetson board and store the profiles as input to the BALB scheduling algorithm.

4.4.2 Impact on Detection Quality

In this section, we evaluate the impact of the proposed framework on the resulted detection quality. Ideally, we want to optimize the neural network processing speed without missing

[3] `https://github.com/ultralytics/yolov5`

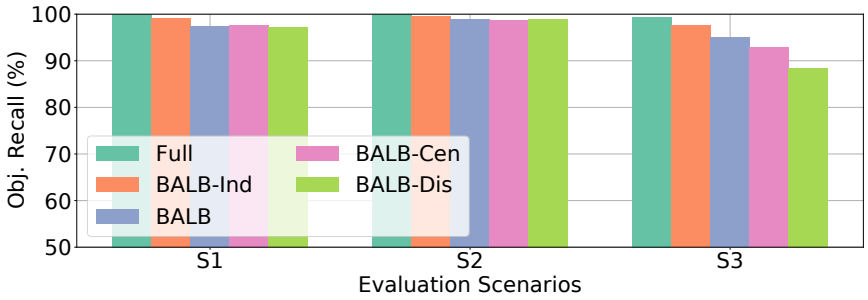


Figure 4.5: Comparisons on object recalls for different algorithms.

any object appeared in the view.

Metric: We use *object recall* as the quality metric here. It is calculated as: At every timestamp, for each groundtruth object, as long as there is at least one camera detects it, then it is counted as a true positive. Otherwise, it is counted as a false negative. The object recall is defined as the ratio between true positives and all groundtruth objects. It is not affected by the missing labels for partially occluded objects.

Baselines: The following baselines are compared:

- *Full*: We perform full frame detection one every frame collected by every camera.
- *BALB-Ind*: Each camera independently runs the BALB framework without considering the spatial correlations among cameras.
- *BALB-Cen*: A variant of BALB that only runs the central stage algorithm.
- *BALB-Dis*: Another variant of BALB that runs the distributed stage to assign objects at every frame.

Analysis: The results are summarized in Figure 4.5. We have the following observations: First, tracking-based slicing almost does not degrade detection recalls, as BALB-Ind achieves similar object recall as full frame detections in all scenarios. Second, through comparing BALB and BALB-Cen, we find that the central stage alone achieves high recall when only a few objects appear at a time (*i.e.*, S1 and S2). However, when many objects appear simultaneously (*i.e.*, S3), there is a degradation on BALB-Cen. This is where we need the distributed stage of BALB. Third, the imperfect correlation model in scenario S3 has a larger impact on BALB-Dis than BALB-Cen. This is because we performed a matching step to associate detected objects and predicted object locations in the central stage, which reduces the false positives in the identified associations. It still remains a challenging problem to implement a fully distributed and self-organized multi-camera analytic system based on their

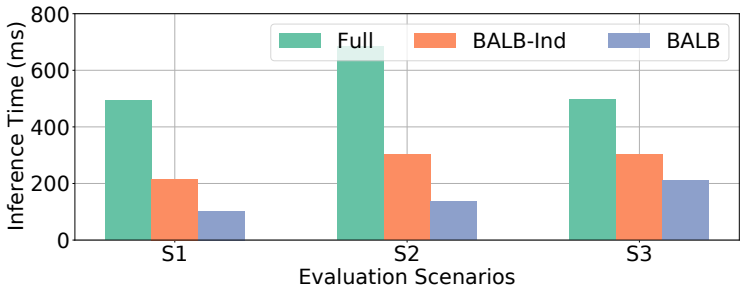


Figure 4.6: Comparisons on YOLO inference time for different algorithms.

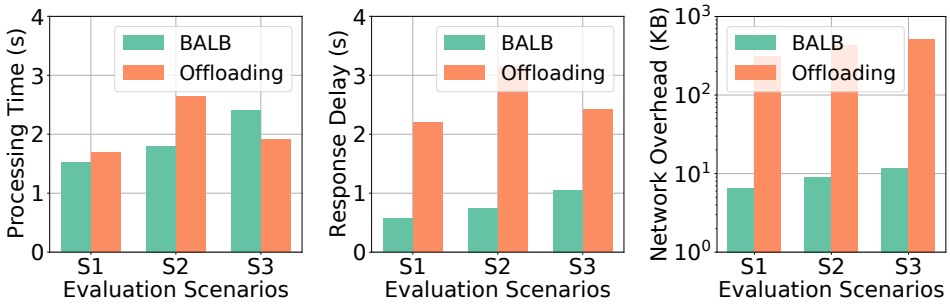
spatial correlations. In conclusion, the complete BALB algorithm provides better quality assurance than using each stage individually.

4.4.3 Neural Network Processing Speed Optimization

We next evaluate the achieved speedup on the inference efficiency. We compare the average per-frame YOLO inference time on the slowest camera for each scheduling horizon. For BALB algorithms, we average the full frame inference time with regular frame times in a scheduling horizon. We compare BALB with full frame detections (Full) and BALB-Ind to see how the spatial and temporal correlations help save the inference time. The corresponding results are presented in Figure 4.6. We can see that BALB-Ind saves more than 50% time compared to full frame detections in the first two scenarios, and saves 47% time in scenario S3 by slicing and batching. Beyond that, BALB further saves 52%, 55%, 30% inference time compared to BALB-Ind in the three scenarios. The saving on scenario S3 is relatively small because more objects appear simultaneously in S3, so after slicing we generate a lot of partial frames. Besides, the cross-camera view overlaps are also smaller compared to the other two scenarios. Putting them together, we attain multiplicative speedups of 4.78×, 4.97× and 2.35× on BALB compared to full frame detections in the three evaluation scenarios. It is worth mentioning that the speedup is larger in applications where we only want to track a few objects by the multi-camera system.

4.4.4 Onboard Processing vs. Cloud Offloading

In this part, we compare the processing time, response delay and network overhead between BALB and MPEG offloading. MPEG offloading is adopted by many centralized video analytics systems [118, 129]. In this approach, we first use H.264 [130] to compress the images of a scheduling horizon into a video, and then upload the video to the server for centralized object detection. We use an RTX 2080 Ti GPU at the server side. Resolutions are same as



(a) Processing time. (b) Response delay. (c) Network overhead.

Figure 4.7: Comparisons between BALB and MPEG offloading on processing time, response delay and network overhead for a scheduling horizon.

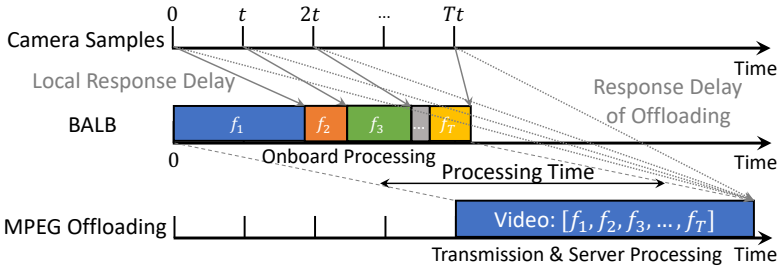
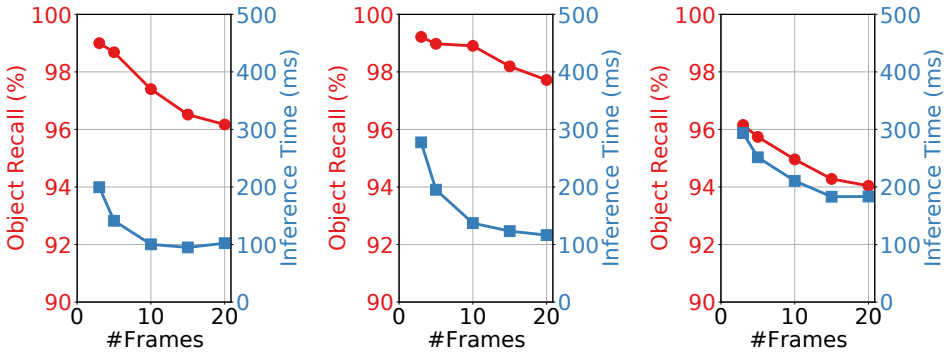


Figure 4.8: Difference between the response latency of BALB and MPEG offloading under the same processing time. $f_1, f_2, .., f_T$ represent frames in a scheduling horizon. BALB sequentially generates detections on sampled frames, while the MPEG offloading only gets the detection results after receiving the response from the server.

what we use in BALB. The results are given in Figure 4.7. *Processing time* is the end-to-end time we need to process one scheduling horizon. BALB shows better speed than MPEG offloading in first two scenarios but is slower in scenario S3. However, their response latency is different by design. *Response latency* is defined as the time between when the frame is captured and when its detections are generated. Video-based offloading naturally produces longer response delay, as illustrated in Figure 4.8. Therefore, BALB is better at providing real-time responses under similar processing speed. In addition, the network overhead (*i.e.*, sum of packet sizes) of BALB is 1-2 orders lower than MPEG offloading.

4.4.5 Impact of Scheduling Horizon Length

In this section, we investigate the impact of the scheduling horizon length on the attained object recall and frame neural network inference time. In Figure 4.9, we plot the change of object recall and inference time w.r.t the number of frames in a scheduling horizon. The observation is that longer scheduling horizons leads to higher inference speed, because the penalty of full frame detections are distributed among more frames, but they also suffer



(a) S01 (b) S02 (c) S03

Figure 4.9: The impact of scheduling horizon length on object recall and YOLO inference time.

Table 4.2: Breakdown Per-frame Latency Overhead

Scenario	Central Stage	Tracking	Distributed BALB	Batching	Total
S01	2.59 ms	18.90 ms	0.08 ms	7.53 ms	29.10 ms
S02	1.11 ms	21.43 ms	0.09 ms	13.21 ms	35.84 ms
S03	2.27 ms	11.55 ms	0.22 ms	19.86 ms	33.90 ms

from lower recalls, which is caused by the inaccuracy of camera correlation models and the tracking algorithm. On the contrary, short scheduling horizons attain higher object recalls at the cost of higher inference time. Choosing the scheduling horizon $T = 10$ achieves a good tradeoff between the detection quality and inference efficiency.

4.4.6 System Overhead

Last, we report the system overhead produced by the components in our framework. The results are summarized in Table 4.2. For each component, we first compute the maximum overhead among cameras at each frame, and then compute the mean overhead across frames. Since the central stage is only called once per scheduling horizon, we distribute its overhead to every frame. The central stage overhead includes both cross-camera object association and central BALB scheduler. The optical flow estimation is not blocked by any other steps, so we put it in a separate thread to run in parallel, such that its computation (11.79 ms on Jetson Xavier, 19.50 ms on Jetson TX2, and 24.80 ms on Jetson Nano) does affect the main thread efficiency. The resulted overhead per frame is between 29.10 ms and 35.84 ms. We can see the overhead mainly comes from the tracking and task batching, which may be further optimized by putting more engineering effort. We leave it as a future investigation.

4.5 RELATED WORK

In this section, we briefly review the recent literature on video processing systems. Most early work [121, 131, 132, 133] on this topic focused on optimizing query-based video analytics. A query is provided and the system needs to automatically find all related information in a large-scale video database. They either optimize the indexing policy and storage structure for video data [131], or reduce the searching and querying effort by filtering out unrelated information [132].

Recent attention [119, 129, 134, 135] was paid to live video analytics systems, where the system needs to coordinate deployed cameras to perform query-based tasks, or general detection tasks, on the live video streams. These systems work in a traditional client-server architecture, where the cameras send the sampled image frames to the cloud for centralized processing. The key idea in this thread is to minimize the amount [119, 134, 136], resolutions [129], or the regions [135] of frames to be transmitted, because the network bandwidth is the main bottleneck in this pipeline.

With increasing compute power on smart cameras, one can provide a better real-time response if neural network processing was done locally. Existing work that utilizes camera compute power [137, 138, 139] focused on partitioning the workload between camera and edge/cloud servers. This work, to the best of our knowledge, is the first effort to optimize the neural network processing speed of a live video processing system purely at the edge devices, by exploiting the spatial-temporal correlations in the multi-view video streams and task batching mechanism on modern GPUs. It achieves similar end-to-end processing time as cloud offloading approach, but outperforms with lower latency in responses and consumes much less network bandwidth. It is thus a better fit for latency-sensitive video analytics applications or deployment scenarios with restricted network connections.

4.6 LIMITATIONS AND DISCUSSION

In this section, we briefly discuss the existing limitations and issues in our multi-camera scheduling work. First, we currently only consider the multi-camera perception scenario with overlapped views, which only occupies a small portion of the real-world deployment situations. More frequently, the deployed cameras may not have direct overlaps in their views at the same time, but such overlaps happen more frequently at the time dimension. For example, the object appears in one camera may appear in another camera in a short time interval. Imagine we are working with the event-based cameras that are normally turned off, we can actually take advantage of their temporal correlations in object appearance to

activate the cameras in advance, to resolve the impact of the initial period in event-based camera activation. Second, we allocate all computations to the camera side, which may increase the hardware cost of the camera platforms in the future. How to appropriately combine the limited compute power at the camera platforms and the strong compute power at the edge/cloud servers, without consuming too much network bandwidth, is the ultimate question we want to answer in the future. Using such an approach, we could achieve a better tradeoff between the onboard processing cost and the network transmission cost in a distributed inference paradigm.

CHAPTER 5: ATTENTION-BASED MULTI-SENSOR FUSION

5.1 OVERVIEW

In several multi-sensor application contexts [140, 141, 142, 143, 144, 145], deep learning algorithms have shown non-trivial accuracy improvements over conventional feature-engineering-based machine learning methods [39, 146], motivating a closer look at the use of deep neural networks for multisensor data fusion [147]. Recent work developed novel neural network architectures for sensor data processing [45, 148, 149, 150], and neural network model reduction techniques for resource-constrained Internet-of-Things (IoT) devices [51, 151, 152, 153]. A key advantage of deep-learning-based solutions over the plethora of model-based approaches lies in reducing the human design burden. In a world increasingly dominated by data and computing power, trade-offs that replace human effort with machine-centered albeit data-intensive approaches are becoming increasingly attractive.

Personal devices, such as smart phones, smart watches, and fitness trackers, are typically equipped with multiple sensors that can be collaboratively utilized to capture user context, perform environmental measurements, and recognize body movements. Several devices may coexist in a typical body network. However, not all devices and modalities are equally useful for detecting a given class of outputs at all times. For example, when detecting walking, a wrist watch or a fitbit would usually be very helpful. However, when wrists are confined, say, by pushing a cart in a grocery store, a cell-phone in a back pocket might work better. How can one automatically decide, based on current measurement features, where to pay attention in a given situation to detect a given class of activity? This automatic attention guidance mechanism is the topic of the work.

The work extends traditional ensemble methods by understanding global context in which given local sensor outputs may be "misleading". For example, the smart watch and the fitbit on the wrists of a person pushing the shopping cart might generate high confidence outputs saying the person is sitting in a slowly moving vehicle. If the only other device on the person correctly detects walking, it might be out-voted. An attention mechanism, in contrast, can attenuate the false claims (despite their associated high local confidence) because the global context suggests that these sensors are presently not in a position to yield accurate local results.

Attention mechanisms are an emerging technique for dynamically adjusting neural network's focus by scaling the features using corresponding weights computed by a separate attention module. Given an appropriate attention design, the network can automatically

amplify the influence of informative features and suppress unrelated noise. In conventional attention mechanisms for regression problems, the extraction of features weights are often computed from matching features from the input signal and features from the output signal. The features used to estimate input features importance are called the *query*. For example, in neural machine translation problems, the word embeddings in the output sentence are used as the query to find alignment with each input sentence word. How to design a query (that can accurately identify informative features) in classification problems with no output signal but just an output label is a key challenge in attention mechanism research.

In this work, we follow an idea called *self attention* proposed by Vaswani *et al.* [2] where the query is also extracted from the input features. Our insight is that we try to match the neural network node outputs (at certain layers) against inputs at earlier layers with the idea that inputs that are more correlated with outputs deserve more attention. We propose a *global attention module* that (i) uses *high-level node features* (*i.e.*, features of nodes that are closer to the output layer) to estimate the contribution (and hence, scaling factor) of each input feature vector in the low-level fusion layer, then (ii) adds these scaled local features to the output (high-level node) features. The design literally superimposes selected local context (scaled low-level features) and global context (high-level node features), allowing subsequent nodes to consider the combination of the two. By adding local context and global context, more informed decisions can be made based on the combination. For example, the system might learn that when a hip-mounted device detects features compatible with walking, while multiple wrist-mounted devices detect features compatible with sitting or standing, the latter devices should be suppressed, as the ground truth is usually more correlated with the former. The attention module is an automated mechanism to learn such global feature weighting policies.

The idea of using information at higher layers to guide attention comes from the commonly observed fact [154] that, in a deep neural network, lower level features are local to the input and general to the task, whereas higher-level features are global to the input but specific to the particular output class. In other words, the high-level features contain more information related to specific output classes. Our attention module design is successfully integrated into an end-to-end learning framework, called GlobalFusion, where two global attention modules, named global position attention and global modality attention, are deployed to fuse information from heterogeneous sensing modalities and diverse sensor positions, respectively. Figure 6.1 provides an overview of the proposed GlobalFusion framework. Our backbone network is based on DeepSense proposed by Yao *et al.* [148]. Actually, global attention is a flexible and configurable module that can be used as building block in most state-of-the-art sensing data processing frameworks when fusing heterogeneous information. The global

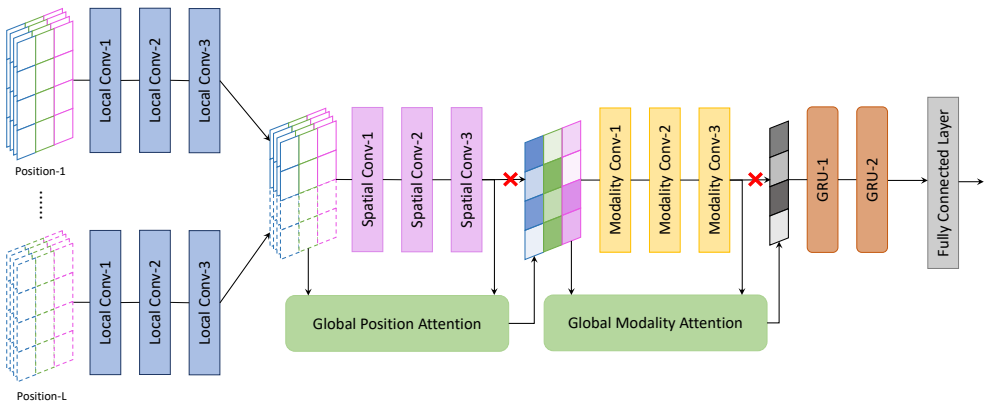


Figure 5.1: GlobalFusion framework overview. (We use different dash lines to represent features from different sensor positions, and use different colors to represent features from different sensing modalities.)

attention module is implemented by a lightweight shallow convolution module, so that its incurred computational overhead is low.

Our design is targeted at general IoT applications because we do not rely on any application-specific insight. Rather, we use a pure data-driven approach. However, due to the limitations of current publicly available sensing datasets, we demonstrate the effectiveness of GlobalFusion on four public human activity recognition (HAR) datasets: 1) PAMAP2 [155], 2) Realworld-HAR [156], 3) DSADS [157], and 4) DG [158]. In these datasets, we are given multiple types of sensor readings collected from several body positions to infer the human activities. We compare GlobalFusion to both the state-of-the-art non-attentional DeepSense [148] framework, and attentional frameworks, including SADeepSense [159], BANet [160], and attnLSTM [161]. GlobalFusion is able to consistently outperform these algorithms with a clear margin on all datasets. Through a qualitative analysis of attention weight distribution, we also demonstrate the interpretability of global attention design. Finally, we test the inference time and energy consumption of GlobalFusion on two commodity IoT devices, Nexus 5 and Raspberry Pi 3 Model B. The results show that the overhead of our global attention module is negligible compared to the backbone DeepSense network.

5.2 MOTIVATION AND CHALLENGE

The goal in this work is to design an information fusion mechanism for human activity recognition (HAR) to successfully utilize information gathered from heterogeneous sensors and diverse body positions. Both effectiveness and efficiency should be considered and well addressed in the fusion design. On one hand, effectiveness means that the fusion mecha-

nism should be able to extract compatible, correlated and complementary information from heterogeneous sensors and diverse body positions, so that the aggregated information at decision level is maximized. On the other hand, efficiency requires the fusion mechanism to be automatic and lightweight in computation. We prefer to avoid complicated human-crafted feature extraction or iterative searching. In conventional human activity recognition frameworks, two representative fusion mechanisms have been widely adopted, input level fusion and decision level fusion. However, as we will show next, neither of the two approaches can completely meet our two objectives mentioned above.

Input Level Fusion. This thread of approaches is also known as signal level fusion [162], where data from multiple sources are combined into higher quality data with better understanding of the environment, before being fed into activity classification models. For example, the ArmTrack system proposed by Shen et. al. [142] utilize the compass, accelerometer, gyroscope data to recover the static human gesture at each time step. From the physical aspect, human gesture is defined as the position of 8 major joints of human skeleton, while human activity is defined as the continuous moving patterns of these joints. In the second stage, they feed the recovered human gesture sequence into the classification model to perform human activity recognition. Although this approach is straightforward and matches well with human intuitions, it has two shortcomings: first, this method requires large amount of existing physical knowledge and skeleton structure theories from biomedical field in gesture recovery. Second, the gesture recovery stage is time consuming because a lot of effort is putting into searching the best gesture to match collected sensor readings. Similar issues exist in most input level fusion mechanisms, because complex computations are needed to generate human-crafted features for model inference. Instead, a better philosophy is to directly feed the heterogeneous input (after possible simple preprocessing) into the learning framework to infer the target human activities. In summary, although being intuitive, the input level fusion mechanisms are too weak at efficiency.

Decision Level Fusion. This approach typically refers to the ensemble learning in machine learning field. We train a separate model utilizing the data from each individual sensor. The training process of each individual model is independent and the training objective is to maximize the individual performance of each model. During the inference, the prediction results by each individual model are aggregated by the corresponding ensemble method, *i.e.*, the voting method. The obtained voting result is used as the ultimate classification result. However, the heterogeneity among sensors not only comes from their noise level caused by different calibration level, sampling rate, or sensing quality; but also comes from their heterogeneous physical intrinsics. The ensemble learning is only good removing the noises and stabilize the model, but unable to extract complementary information from heterogenous

sources. The major drawbacks of ensemble learning lies in two aspects: first, the training process of each individual model is independent to each other, which means there is no interactions or collaborations across sensors involved in the training. Each individual model is trained to maximize the information it contained related to target activities. However, the aggregated information after voting is not necessarily maximized, because a lot of overlapped information is included. Second, the ensemble learning is not flexible in information fusion. Although complicated voting rule can be designed and used in ensemble inference, its flexibility is still quite limited. In human activity recognition, the weights among different sensors and body positions can be dynamic at different time, because their corresponding activities are different. For example, sensors at fore arm are more useful in discriminating between washing hands and opening door, while sensors at legs are more critical in classifying walking and running. Thus, we can not predefine a voting rule in advance. Moreover, since the sensing data is a time-series representation, the importance of different sensors can be even different at different time steps within one sequence. Ensemble methods are apparently incapable of providing such flexibility. In short, the effectiveness of decision level fusion is far from satisfactory.

Therefore, we try to solve this problem with recent advancement of attention mechanisms in deep learning. It is an emerging technique for dynamically adjusting neural network model's focus by multiplying the features of each sensor with a corresponding weight, where the weight is computed dynamically during runtime. With appropriate attention design, the network can automatically amplifies the influence of informative features and suppress unrelated noises. The information fusion is performed at the intermediate feature space. No complex data recovery algorithm or voting strategy design is needed. The computation of the attention weight is based on a separate small neural network module, where the specific structure can be selected. The attention module is trained together along with the main network in an end-to-end manner. Therefore, the computation is totally automatic from multi-sensor data to predicted classes, instead of partitioned into two stages as input level fusion mechanisms. Consequently, a great tradeoff between effectiveness and efficiency can be achieved by attentional neural network models. We will introduce the design details of our framework in next section.

5.3 GLOBALFUSION FRAMEWORK

Our idea is motivated by the commonly observed fact [154] that in a deep neural network, the lower level features are local to the input and general to the task, while higher level features are global to the input and specific to particular classes. We call the feature

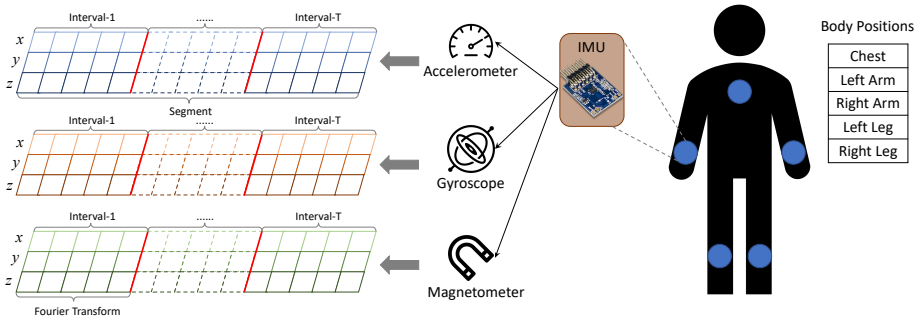


Figure 5.2: An example illustrating our data preprocessing and input format.

vectors at lower layers as *local features*, while call the feature vectors at higher layers as *global features*. Therefore, juxtaposing local features and global information can help improve classification based on the combination. In this section, we first describe the general architecture of GlobalFusion, which is an end-to-end deep learning framework designed for multisensor information fusion. Walking through the GlobalFusion architecture, we point out the positions in network where we need a global attention module and outline requirement for its correct functionality. Next, we explain the technical details of global attention module design for multisensor information fusion, especially how to utilize high level global features to compute low level attention weights.

Before diving into the details, we first introduce the notations used in the rest of this work. All vectors are denoted by bold lower-case letters (*e.g.*, $\mathbf{x}$ and $\mathbf{y}$), while matrices and tensors are represented by bold upper-case letters (*e.g.*, $\mathbf{X}$ and $\mathbf{Y}$). For a vector $\mathbf{x}$, the j^{th} element is denoted by $x_{[j]}$. For a tensor $\mathbf{X}$, the t^{th} matrix along the first axis is denoted by $X_{[t\cdot\cdot]}$, and other slicing denotations are defined similarly. Assume we have L spatial positions and S distinct sensor types deployed at each position, $\mathbf{X}^{sl}$ means the s-th sensor reading at l-th position. For each layer k, we use $\mathbf{X}^{(k)}$ to represent the input to this layer, and use $\mathbf{Y}^{(k)}$ to denote corresponding output. For any tensor $\mathbf{X}$, $|\mathbf{X}|$ denotes the size of $\mathbf{X}$.

5.3.1 GlobalFusion Architecture

Before introducing the technical design of global attention, we first give an end-to-end overview of the GlobalFusion framework for multisensor information fusion. GlobalFusion is based on the state-of-the-art DeepSense[148] framework as back-bone network. It exploits the power of both Convolution Neural Networks (CNN) and Recurrent Neural Networks (RNN) in time-series sensing data processing. In the vanilla DeepSense model, fusion of musitisensor inputs is performed by a three-layer convolution module after concatenating

inputs from all fusing components. By doing so, the model does not take the heterogeneity among input sources into consideration and pays equal attention to each fusing component. Thus, the fusion layer cannot maximize the information extracted from all information sources (*i.e.*, sensing modalities and body positions). To solve this problem, we cut the direct connection between output of information fusion convolution module and the input of the next layer, and add one global attention module between them to first enhance fusion output with complementary local features before feeding it into next layer. The overall architecture of GlobalFusion is presented in Figure 6.1. In this subsection, we temporarily regard the global attention module as a black-box implementation which is able to automatically extract complementary information from each input local feature vector compared to the output global feature vector. The information fusion at sensing modalities and diverse body positions are both included.

Suppose the sensing data $\mathbf{X}$ comes from L spatial locations, where N sensors are deployed simultaneously at each position. This models the general scenario of intelligent human activity recognition where multiple integrated sensing units such as Inertial Measurement Units (IMUs) are deployed at a set of diverse body positions to carry out the sensing task collaboratively. Each sensing unit consists of multiple sensors. For example, most IMUs usually contain accelerometer, gyroscope, and magnetometer so that the moving speed and orientation can be simultaneously tracked. For a given sensing modality n and spatial location l, its sensor readings are divided into fixed-length but non-overlapped time segments. One segment aggregated from all spatial positions and sensors constitutes one data sample in our model. Each sensor also has d dimensions (*i.e.*, x, y, z axises). Number of dimensions can be different among sensors, but we assume the same dimension just for notation simplicity. Different sensor readings are upsampled and downsampled into a unified sampling rate. Each data segment is further divided into T non-overlapped time intervals. In data preprocessing step, we perform a Fourier transform to each interval to extract their frequency domain representations, which have been proved to be more informative than pure time domain representations [163]. By utilizing the time-frequency input, both the time domain order information and frequency domain pattern information are well preserved. To help illustrate our data segmentation and pre-processing procedure, we show an example of data input to our model in Figure 5.2. After preprocessing, the input fed into the model should have a shape of $\mathbf{X} \in \mathbb{R}^{T \times N \times L \times d \times 2f}$, where T represents time intervals, N denotes sensors, L denotes spatial locations, d is the sensor dimension, and $2f$ is spectral samples with f frequency magnitude and phase pairs within each interval. The order of dimensions in $\mathbf{X}$ is determined by the order of information fusion in GlobalFusion.

In general, GlobalFusion is divided into four stacked sub-modules: individual sensor con-

volution module, spatial fusion module, modality fusion module, and time recurrent module. We will introduce these sub-modules one by one from bottom to top.

Individual Convolution Module. The processed sensing data from each (sensor, position, interval) combination is first separately fed into the individual convolution module. The input data is $\mathbf{X}^{sl}_{[t\cdot\cdot]}$, where s denotes sensor, l denotes position, t denotes time interval. No sensor or body position interaction is considered in this module. Convolution layer [164] has been successful in aggregating information from local area, *i.e.*, adjacent frequencies in our problem. In GlobalFusion, convolution layers are used to gradually extract frequency pattern features within the spectrum of each time interval. We call the output of this module as *(sensor, position, interval) features*, which contain the information contained in given sensor at a specific body position and time interval. They are also the input of spatial fusion module.

Spatial Fusion Module. In this module, we fuse the obtained local (sensor, position, interval) features for same sensing modality across different spatial positions, into *(sensor, interval) features.* (sensor, interval) features represent the information contained in given sensor across all body positions at specific interval. Regarding the information fusion order, we profiled the performance improvement of spatial-fusion-first strategy and modality-fusion-first strategy. It turns out that spatial-fusion-first models consistently outperform modality-fusion-first models, so we choose to first perform spatial fusion in GlobalFusion. We will give more explanation on this design choice from the heterogeneity level in Section 5.4.6. We preserve the three-layer convolution module in DeepSense [148] to extract preliminary global information from different body positions. However, all body positions are homogeneously convolved by now so that some local information is inevitably missed while some noises are included. After that, we stack a global position attention module to allow the incomplete global information to absorb more complementary information from input features at each spatial position. These two sub-modules together constitute our spatial fusion module. The input to this module is $\mathbf{X}^{sl(4)}_{[t\cdot\cdot]}$ from sensor s across every spatial position l, where upper subscript (4) means input to the 4-th layer. The output is fed into next-level modality fusion module for further information aggregation across all sensing modalities.

Modality Fusion Module. In this module, we fuse (sensor, interval) features $\mathbf{X}^{s(7)}_{[t\cdot\cdot]}$ collected from all sensing modalities s into an *interval feature* vector, which is a general feature representation of time interval t. Specifically, we still use a three-layer convolution module to extract preliminary global information from all sensing modalities, the output of which contains incomplete interval features. We use another global attention module to help incomplete interval features absorb more complementary information from input (sensor, interval) features. Deploying global fusion module here is more beneficial than deploying it

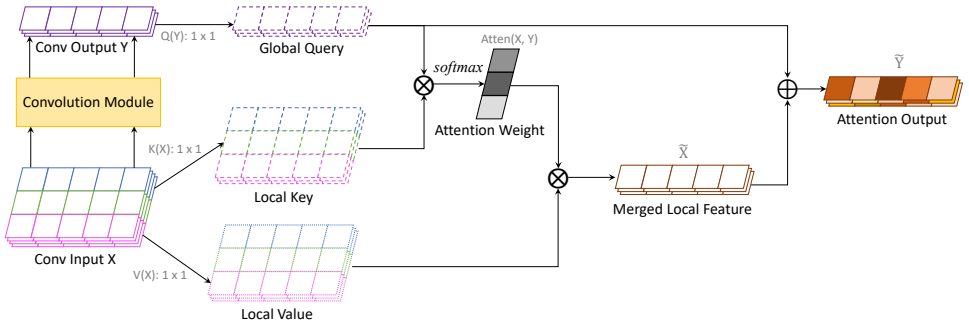


Figure 5.3: Global attention module design. (We use different colors to represent input features from each fusing component. $\bigotimes$ represents dot product operation, while $\bigoplus$ represents element-wise addition operation.)

at spatial fusion level for two reasons: first, the heterogeneity level is higher among different sensing modalities compared to different spatial positions of the same sensing modality; second, modality fusion layer is closer to the output layer, which means the global features at this level is more class-specific and "mature" according to the observations in [165]. After the post-processing by global attention, we feed the flattened vector representation of each time interval into next level time recurrent module.

Time Recurrent Module. After obtaining each interval feature vector, the activity recognition problem turns into a typical sequence classification problem. Here we use a stacked two-layer Gated Recurrent Unit (GRU) to sequentially encode the temporal pattern information. The input to this module is the interval features $\mathbf{X}^{(10)}_{[t\cdot\cdot]}$ across all time intervals t. The order information across time intervals is extracted by the recurrent neural network. We do not use any attention mechanism here for two reasons: first, the step-by-step recurrent structure has already been excellent enough to learn the global temporal patterns; second, it is difficult to decide attention weight of input at each time interval since the information contained in hidden state of each interval is an aggregation of all previous intervals. We simply take the average of hidden states at all time intervals as the output, and feed it into the last fully connected layer for ultimate class prediction.

After obtaining a high-level understanding of GlobalFusion architecture and the corresponding deploying position of global attention modules, we are going to discuss more technical details about the global attention design in next subsection.

5.3.2 Global Attention Module

We explain the specific design details of our global attention module in this subsection. Suppose there is already a feature extraction module in the backbone network that can take

input from multiple input sources and output a feature vector containing information aggregated from all sources (*i.e.*, sensing modalities or body positions). For example, we use the three-layer convolution module in DeepSense [148] to serve this purpose. However, in these modules, the heterogeneity among sources is not necessarily addressed and their information is homogeneously merged. We ultimately want to see the features from informative sensing modalities/body positions being amplified, while the unrelated noises being suppressed. To narrow down the gap between our learning objective and limitation of backbone feature extraction modules, the global attention mechanism is accordingly designed. In global attention module, the complementary information is extracted from input local features and added to global output features before feeding them into next layer.

It has been widely accepted in machine learning community that for a deep neural network, the features of lower layers are specific to the input but general to the classes, while the features of higher layers are general to the input but specific to the classes. Let's use *local features* and *global features* to refer to input and output of the feature extraction module respectively. The global features possess more class-related information than the local features. Therefore, we use the global features as the *global query* to estimate the importance, *i.e.*, attention weight, of each sensing modality/body position. By doing so, the class-specific information in global features become the standard to evaluate the informativeness of local features. During the back-propagation, the class-related information is propagated from the output layer to the attention module, further back to the global/local feature extraction part. Different from the design in [159], where the mean of all input features is used as the query, our global attention design handles the heterogeneity among input sources better because we do not assume that the information from all sources is similar. The residual connection between global query and highlighted local features can help the global features absorb more complementary local information. A graphical illustration of our proposed global attention module is given in Figure 5.3.

As mentioned before, in GlobalFusion, the preliminary global feature extraction module refers to a three-layer convolution network (*i.e.*, the yellow box in Figure 5.3, but it can be configured as other well-behaved computation units). Suppose the input and output of this unit are $\mathbf{X} \in \mathbb{R}^{N \times W \times C_1}$ and $\mathbf{Y} \in \mathbb{R}^{1 \times W \times C_2}$ respectively, where N is the number of components to be fused (we denote it as height of features in the figure), W is the width of features, C_1 and C_2 represent input channels and output channels separately. $\mathbf{X}$ and $\mathbf{Y}$ can be of any shape, we use vectorized local features here just to simplify the notation and make it visually easy understanding. What we need is an attention function $Atten : (\mathbf{X}, \mathbf{Y}) \rightarrow w \in \mathbb{R}^{1 \times N}$ that takes $\mathbf{X}$ and $\mathbf{Y}$ as input to calculate the attention weight of each input source automatically

so that we can fuse the multisensor inputs correspondingly as follows:

$$\widetilde{\mathbf{X}} = \sum_{i=1}^{N} Atten(\mathbf{X_i}, \mathbf{Y}) \cdot \mathbf{X_i}, \quad \text{s.t.} \quad \sum_{i=1}^{N} Atten(\mathbf{X_i}, \mathbf{Y}) = 1. \tag{5.1}$$

For the design of attention module, we adopt the standard multiplicative attention mechanism [2], where the dot product between the query and keys are used to estimate the attention weight for each sensor. Instead of proposing a new attention computation paradigm, the key innovation lies in the choice of the query. Instead of computing the compatibility score between each local feature vector $\mathbf{X_i}$ and global representation $\mathbf{Y}$ directly, we first let them go through a transformation independently. There are two reasons for this pre-transformation: first, the local features and global features belong to different latent spaces, so a non-linear projection can transform them into the same latent space to make them compatible in semantic level; second, the dimensions of local features and global features can be different in general case so that they are mathematically incompatible, while this transformation can unify their dimensions by needs. Adopting the concepts in [2], we call the transformation for global features and local features as *query function* $Q(\mathbf{Y})$ and *key function* $K(\mathbf{X})$ respectively. What differs from previous works is that, instead of extracting both keys and query from local features, we choose to use global output features to extract query, which is closer to the output layer to provide more global information. Both functions are chosen as 1×1 convolution with *relu* activation followed by a flatten operation. After the transformation, both local keys and global query have been projected into a h dimension latent space, *i.e.*, $K(\mathbf{X}) \in \mathbb{R}^{n \times h}$ and $Q(\mathbf{Y}) \in \mathbb{R}^{1 \times h}$. h is a hyper-parameter that we can adjust during training. According to our experience, model performance is not sensitive to the value of h, so this value is empirically fixed at 64, in all 4 datasets during evaluation. Next, we use projected keys and query to calculate their compatibility score, *i.e.*, attention weight for each fusing component:

$$Atten(\mathbf{X}_i, \mathbf{Y}) = \text{softmax}\left(\frac{K(\mathbf{X_i})^T Q(\mathbf{Y})}{\sqrt{h}}\right). \tag{5.2}$$

Here we choose the dot product similarity (*i.e.*, multiplicative attention), instead of a feed-forward neural network (*i.e.*, additive attention), as the compatibility function to compute the attention weight, because it is more intuitive and computational efficient. We scale the product by a factor of $\frac{1}{\sqrt{h}}$ to prevent it from becoming too large to get into regions with extremely small gradients after applying outlier *softmax* [2]. The outlier *softmax* is used to emphasize informative features, and normalize the attention weight distribution. By far, each input source corresponds to a normalized attention weight. Next, we let the local features go

through another transformation, called *value function* $V(\mathbf{X})$, to match their dimensions with the global query. The local values are multiplied with their attention weights and sumed up. The weighted sum is called *merged local features*. After that, we combine the merged local features with the global query $Q(\mathbf{Y})$ through a residual connection later. The *value function* $V(\mathbf{X}) \in \mathbb{R}^{n \times h}$ consists of a 1×1 convolution operation with *relu* activation and a flatten operation. The merged local features are computed by:

$$\widetilde{\mathbf{X}} = \sum_{i=1}^{N} Atten(\mathbf{X_i}, \mathbf{Y}) \cdot V(\mathbf{X_i}), \tag{5.3}$$

where $\cdot$ means that the scalar attention weight is propagated to each element of local feature vector. This merged local feature vector provides a good complement to global feature (not replacement), so we combine them up through a residual connection:

$$\widetilde{\mathbf{Y}} = Q(\mathbf{Y}) + \widetilde{\mathbf{X}}, \tag{5.4}$$

which becomes the output of our global attention module. To fit it back into original network dimension, we can let the output go through another 1×1 convolution for dimension extension or suppression if necessary.

From a different perspective of view, our global attention can be regarded as an innovative residual connection design [19] for sensing data processing. Instead of directly connecting the input and output of a processing module, we first utilize the calculated output to search and highlight local features in input before merging them. The heterogeneity in local features is well addressed while at the same time the residual property is preserved. So far, we have introduced all technical details related to GlobalFusion framework and global attention module design. Next, we validate the effectiveness of GlobalFusion especially the contribution of global attention modules through experiments on four realworld human activity recognition (HAR) datasets.

5.4 EVALUATION

In this section, we compare GlobalFusion to other state-or-the-art deep learning frameworks using four publicly available human activity recognition (HAR) datasets. We first introduce the experimental setup, datasets used, data preprocessing steps, and baseline algorithms we are comparing with. We then show the evaluation results for each dataset, make qualitative observations, and discuss insights obtained from attention weight distributions.

Table 5.1: Statistical Summary of Selected Datasets.

Dataset	Activities	Subjects	Sensors	Positions	Segment	Intervals	Spectral Samples
PAMAP2	18	9	3	3	2 sec	10	20
RealWorld-HAR	8	15	4	5	2 sec	10	10
DSADS	19	8	3	5	5 sec	5	25
DS	2	10	1	3	2 sec	8	16

Finally, we present time and energy efficiency comparisons on commodity IoT devices.

5.4.1 Experimental Setup

All the models evaluated in this work are trained with Tensorflow 1.14 [29] on a workstation equipped with an Intel i9-9960X processor, 64GB memory, and four NVIDIA RTX 2080 Ti GPU. For training the model, we adopt a standard cross entropy loss for classification, along with L2 normalization. The normalization factor is set as 5e-4. The model is optimized by the ADAM algorithm [166] with a learning rate of 1e-4, while $\beta_1 = 0.5$ and $\beta_2 = 0.9$. We add a batch normalization layer and a dropout layer after each convolutional layer to stabilize the training process and prevent overfitting. Training batch size is set as 64.

5.4.2 Datasets

We evaluate inference accuracy on human activity recognition tasks that use multiple sensors as input. All models are evaluated under a leave-one-user-out scenario with k-fold cross validation. Specifically, one subject is chosen as the test user each time, while the activity traces of all remaining subjects are used for training. The test user is then rotated until all users have been excluded. A statistical summary of each dataset is listed in Table 6.1.

PAMAP2 Physical Activity Monitoring Data Set (PAMAP2). [155] This dataset contains data of 18 different physical activities (*e.g.*, walking, cycling, playing soccer, etc) performed by 9 subjects using 3 inertial measurement units (IMUs) that are put at the chest, wrist (of dominant arm), and dominant side's ankle respectively. Each IMU records readings from a 3-axis accelerometer, gyroscope and magnetometer. The sampling rates of all sensors are 100 Hz. We divide data into segment of $2s$ where each segment is further divided into $T = 10$ fixed-length and non-overlapped time intervals. Each interval contains 20 spectral samples. Sensor readings in each interval are sent through a Fourier transform as pre-processing. "subject109" is excluded for testing because s/he has too few contributed

data samples.

RealWorld Human Activity Recognition (RealWorld-HAR). [156] This dataset covers 8 activities (climbing stairs down and up, jumping, lying, standing, sitting, running/jogging, and walking) from 15 subjects on 7 body positions. Sensing modalities include acceleration, GPS, gyroscope, light, magnetic field, and sound level. We choose 5 body positions out of 7 (*i.e.*, head, chest, forearm, waist, and shin), and use four sensor types only (*i.e.*, accelerometer, gyroscope, magnetometer, and light). Upper arm and thigh data are not used because we want to make chosen body positions more diverse. GPS and sound level readings are not used here because their sampling rates are too low. GPS is sampled at 0.08 Hz and sound level is sampled at 2 Hz, while all remaining sensors have a $\sim$50 Hz sampling rate. The sampling rate of all selected sensors is interpolated to 50 Hz by up-sampling and down-sampling. We still use segment of $2s$ consisting of 10 non-overlapped intervals, where each interval contains 10 spectral samples. Every subject is selected once as test user in cross validation.

Daily and Sports Activities Data Set (DSADS). [157] In this dataset, each of 19 activities (*e.g.*, sitting, standing, ascending and descending stairs, exercise on stepper, playing basketball, etc) is performed by 8 subjects (4 female and 4 male). The contained sensors are still accelerometer, magnetometer, and gyroscope on 5 body positions (torso, right arm, left arm, right leg, and left leg). The data sampling rate is 25 Hz, so we choose the segment length as $5s$ with 5 intervals within each segment. Therefore, we have 25 spectral samples in each time interval. Every subject is selected once as test user in cross validation.

Daphnet Gait (DG). [158] The daphnet freezing of gait dataset is devised to benchmark automatic methods to recognize gait freeze from wearable acceleration sensors placed on legs and hip. It is a binary classification problem (*i.e.*, freeze or not freeze). Only accelerometer data at three body positions (*i.e.*, ankle, upper leg, trunk) is provided. Here we do a minor adjustment on our GlobalFusion model, where the modality fusion module is removed. It is collected from 10 Parkinson's disease patients. The sampling rate is 64 Hz. We choose a $2s$ segment and divide it into 8 time intervals, so that each interval has 16 spectral samples. Similarly, every subject is selected once as test user in cross validation.

The purpose of using the first three datasets is two-fold. First, we want to compare our model to the baselines on multiple multisensor datasets to reach more broadly substantiated conclusions. Second, we want to find common observations in attention weight distributions to check if attention allocation meets intuition. Through the DG dataset, we want to see whether proper attention mechanisms can overcome the unbalanced class distribution problem that is heavily represented in that dataset.

5.4.3 Baselines

Before presenting the results, we briefly review state-of-the-art deep learning frameworks for heterogeneous information fusion that are selected as baselines in our experiments. We pay special attention to attention-mechanism-based designs.

GlobalFusion-Single: To show the effectiveness of our global attention design and give a straightforward understanding of contribution by each attention module, we choose a variant of GlobalFusion here. In this model, only global modality attention is used before time recurrent layer. Comparing performance of this model with DeepSense could help understand the effectiveness of global modality attention module, while comparison with GlobalFusion could help show the contribution of global position attention module.

DeepSense [148]: This is the back-bone network for our GlobalFusion framework, which is one of the commonly used sensing data processing frameworks for IoT applications. It's generally based on the combination of convolutional modules (*i.e.*, within a time interval) and a recurrent module (*i.e.*, across time intervals) for temporal pattern extraction. To emphasize the effectiveness of our global attention module, we use the same backbone architecture as GlobalFusion here excluding global attention modules.

SADeepSense [159]: This is a recent self-attention based DeepSense framework. They use a self-attention (SA) module for heterogeneous sensor information fusion and time-series information fusion respectively. They use the mean of all input features as the query to estimate correlation across sensors / time intervals. They also adopt the multi-head design to learn the correlations from different latent spaces. In our implementation, three SA modules are deployed at spatial fusion, sensor fusion, and time fusion respectively based on their design in the paper. This design utilizes the local features as the query. Through comparing GlobalFusion and SADeepSense, we can see the advantages of using global features as the query in attention module.

attnLSTM [161]: They use two attentional modules to improve the classification performance of recurrent networks. We make minor enhancement to their framework: at the bottom of network, instead of using raw sensing input, we use a same three-layer convolutional module as GlobalFusion to extract the local features of each individual sensor within each time interval. The reason is that we want to compare the impact of different fusion mechanisms instead of lower-level feature extraction parts. Next lies the sensor attention part, where a shallow fully-connected module works as attention unit across different sensing modalities and sensor positions. Finally, after one LSTM layer, they use the output of last time interval to estimate importance of hidden state at each previous time interval, and take the weighted average of them for final prediction. In addition, they add a continuous

constraint to both attention modules to regulate the attention weight to change smoothly over modalities and time intervals.

BANet [160]: This model exploits a different order of information fusion, where they first fuse information at different time intervals for same sensor, and then fuse information across sensing modalities. We still add an individual convolution module at the bottom of their architecture to extract low-level features. Temporal attention is computed by a 1×1 convolution followed by a softmax layer, which belongs to brute-force additive attention and no compatibility design is considered. Sensor attention is computed in a similar way: they use one fully-connected layer followed by a softmax layer.

5.4.4 Quantitative Classification Results

In this subsection, we present the evaluation results of GlobalFusion and aforementioned baseline frameworks on each selected dataset. We will use abbreviations for models in the tables and figures. GF represents the GlobalFusion with double attention modules, where both global position attention and global modality attention are deployed. GF-SGL or GlobalFusion-SGL both represent the GlobalFusion with global modality attention only. DS is short for DeepSense, while SA-DS is used for SADeepSense. In addition, DS-Acc, DS-Gyro, DS-Mag, and DS-Lig represent DeepSense-Accelerometer, DeepSense-Gyroscope, DeepSense-Magnetometer, DeepSense-Light respectively. Only one sensing modality is used in these individual models.

We mainly use accuracy and macro F1 score as the classification performance metrics, while micro F1 score is not used. In multi-class classification problems, micro precision = micro recall = micro F1 score = accuracy. In this case, true positives (TP) are defined as the samples that were predicted to have the correct label. Every time there is a false positive (FP), there will always also be a false negative (FN) and vice versa, because always one class is predicted. Therefore, the micro F1 score is redundant with accuracy. Instead, the macro F1 score can better reflect the model classification performance across all classes. It prefers more balanced classification results. This is a good complement to accuracy for capturing performance balance across classes. In our tables, "Acc." is short for accuracy, and "Mac. F1" represents macro F1 score.

PAMAP2 Results: We start with the PAMAP2 dataset. In Table 5.2, we give the accuracy and macro F1 score for all models compared, including both individual testing cases and the overall average values. Since the evaluation is performed under leave-one-user-out scenario, the model performance is supposed to be lower than random partition, considering the heterogeneity and transfer difficulty between training users and testing user. From

Table 5.2: PAMAP2 classification result.

User	GF		GF-SGL		DS		SA-DS		attnLSTM		BANet	
	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1
Sub. 1	78.99%	75.61%	77.69%	75.76%	73.53%	69.57%	72.20%	67.43%	72.04%	67.21%	72.61%	69.32%
Sub. 2	92.22%	92.45%	91.75%	91.52%	74.77%	71.10%	83.28%	82.13%	74.84%	71.34%	69.77%	71.42%
Sub. 3	95.37%	75.67%	94.91%	68.70%	94.95%	68.52%	93.87%	62.32%	93.15%	67.41%	89.54%	65.00%
Sub. 4	95.26%	86.70%	95.74%	86.92%	94.02%	93.89%	88.24%	79.86%	88.79%	81.05%	91.91%	76.20%
Sub. 5	92.93%	92.36%	93.23%	92.75%	92.93%	92.67%	90.25%	88.94%	88.62%	87.45%	90.92%	89.79%
Sub. 6	92.88%	84.77%	92.27%	88.36%	90.79%	82.67%	89.72%	87.06%	91.78%	83.19%	85.44%	77.89%
Sub. 7	96.09%	94.65%	96.35%	94.72%	95.75%	94.28%	94.70%	86.26%	94.44%	85.30%	93.58%	84.71%
Sub. 8	83.09%	79.78%	80.05%	77.15%	59.69%	54.98%	63.13%	59.14%	68.86%	62.42%	55.23%	49.67%
Overall	**90.86%**	**85.25%**	**90.25%**	**84.48%**	**84.55%**	**78.46%**	**84.42%**	**76.64%**	**83.69%**	**75.67%**	**81.13%**	**73.00%**

Table 5.2, we can see that DeepSense obtains similar accuracy and macro F1 score as other state-of-the art attentional frameworks, while our global attention module further improves its performance by a clear margin, especially the global modality attention module (*i.e.*, GF-SGL v.s. DS). Compared to modality-attention only GlobalFusion-Single framework, GlobalFusion further improves the accuracy and macro F1 score by a small margin. We can regard it as a trade-off between efficacy and efficiency when considering model deployment on IoT devices. If we want to obtain better model efficiency, GlobalFusion-Single can be used; otherwise, GlobalFusion is a better choice for higher model efficacy. We also notice that SADeepSense does not show clear improvement compared to vanilla DeepSense here. The reason is that mean value of all sensors are used as the query to estimate the attention weight of each sensing modality in SADeepSense, which leads the model to attend to more homogeneous input, so that the model behaves similar as non-attentional DeepSense. Among the users, those with lower accuracies at back-bone DeepSense model (*e.g.*, Subject2 and Subject8) typically have a higher chance to see a larger improvement by attention learning. According to the confusion matrix of GlobalFusion in Figure 5.4 (a), we can see an ambiguity between sitting and standing. This ambiguity among static gestures also exists in RealWorld-HAR and DSADS results. In later analysis, we will show that this results from the dominant effect of accelerometer features that share the same pattern under all static gestures.

RealWorld-HAR Results: The evaluation results on RealWorld-HAR dataset is given in Table 5.3. Compared to other datasets, this dataset covers a wider range of subject diversity, reflected in their age, height, weight, gender, and dominant arm. Therefore, we can see a large difference across different models. attnLSTM is the worst model here. It only achieves an accuracy of 70.83%. The failure of attnLSTM indicates us that incorrect attention design can lead to a severe performance degradation. DeepSense also does not work well on this dataset. The reason is that the reliability of different sensing modalities differs in a large degree in this dataset (*i.e.*, their single-modality performance are quite diverse as we will show in Figure 5.8 later). Thus, equally weighting and merging different

Table 5.3: RealWorld-HAR classification result.

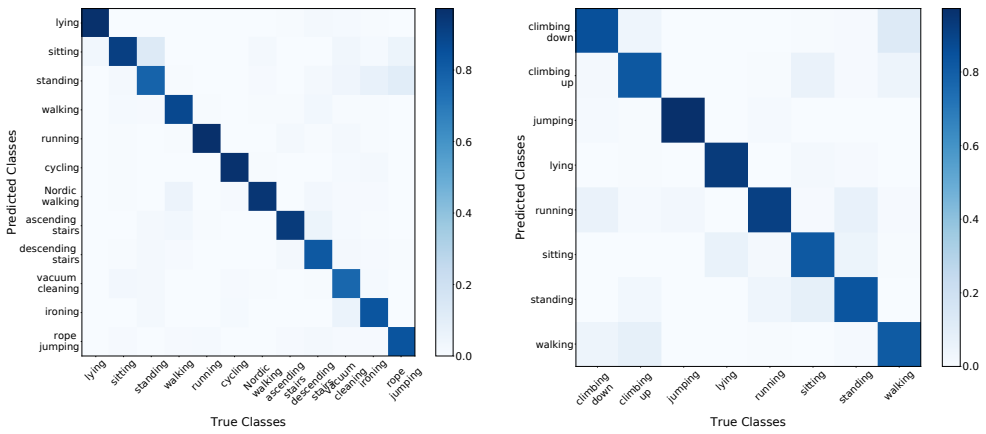
Test User	GlobalFusion		GlobalFusion-SGL		DeepSense		SADeepSense		attnLSTM		BANet	
	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1
Subject1	78.22%	79.36%	75.36%	75.81%	65.95%	65.66%	67.05%	66.31%	65.49%	64.58%	74.95%	74.54%
Subject2	94.39%	94.25%	95.29%	95.57%	80.42%	78.47%	79.20%	78.63%	80.91%	79.58%	93.07%	93.18%
Subject3	78.53%	71.37%	75.05%	62.06%	54.51%	51.91%	54.73%	51.89%	48.53%	42.10%	49.73%	50.41%
Subject4	92.61%	93.38%	91.17%	92.64%	86.62%	88.47%	82.23%	84.08%	89.26%	89.44%	84.18%	85.99%
Subject5	81.63%	81.11%	82.63%	82.22%	72.57%	72.29%	79.52%	78.83%	76.93%	74.89%	68.91%	69.30%
Subject6	84.14%	85.14%	84.47%	85.72%	67.52%	66.90%	73.77%	73.05%	69.51%	67.84%	73.39%	73.54%
Subject7	78.34%	70.88%	80.59%	78.05%	69.56%	70.10%	67.56%	62.77%	78.02%	79.30%	77.16%	78.79%
Subject8	63.54%	66.51%	62.79%	63.05%	39.39%	35.17%	50.08%	48.71%	24.30%	18.01%	47.94%	48.41%
Subject9	91.21%	92.07%	90.76%	91.44%	80.58%	74.77%	77.99%	74.24%	80.31%	77.89%	76.29%	75.64%
Subject10	96.18%	95.84%	96.33%	96.13%	95.07%	94.37%	85.35%	85.71%	80.27%	78.00%	82.76%	83.03%
Subject11	89.13%	89.82%	86.87%	87.46%	85.34%	85.92%	78.49%	76.16%	76.19%	74.54%	79.14%	77.95%
Subject12	95.12%	95.36%	95.88%	96.01%	83.85%	82.04%	85.70%	85.29%	86.46%	86.45%	81.72%	79.93%
Subject13	88.45%	89.48%	88.73%	89.68%	71.88%	73.51%	75.60%	75.19%	78.77%	80.13%	73.21%	73.83%
Subject14	90.51%	72.83%	83.62%	84.91%	71.82%	69.77%	75.06%	73.68%	50.45%	48.07%	79.85%	78.77%
Subject15	89.58%	89.61%	90.40%	90.42%	81.30%	79.92%	78.13%	77.34%	77.11%	76.99%	73.07%	73.82%
Overall	**86.11%**	**84.47%**	**85.33%**	**84.74%**	**73.76%**	**72.62%**	**74.03%**	**72.79%**	**70.83%**	**69.19%**	**74.36%**	**74.48%**

Table 5.4: DSADS classification result.

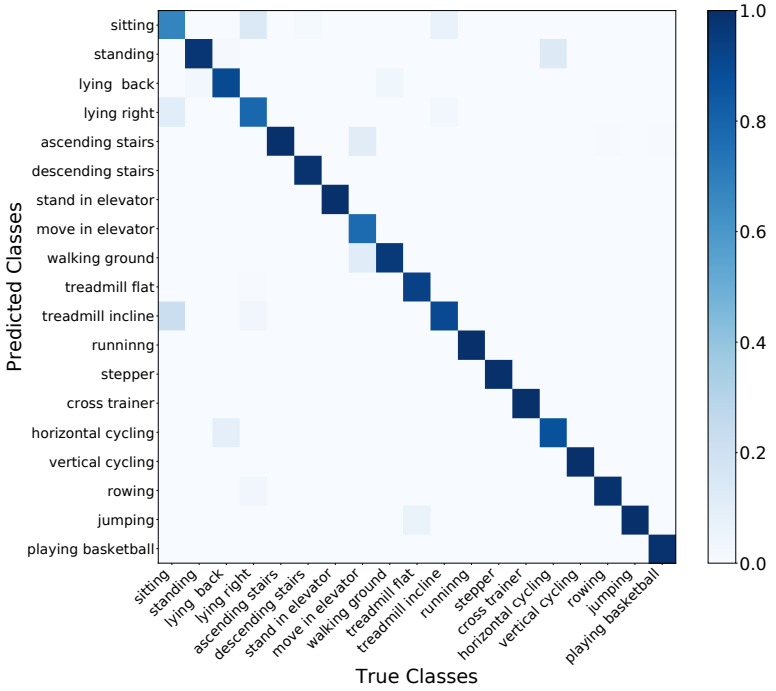
Test User	GlobalFusion		GlobalFusion-SGL		DeepSense		SADeepSense		attnLSTM		BANet	
	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1
Subject1	91.67%	90.63%	91.67%	90.58%	89.06%	87.51%	91.82%	90.73%	90.07%	88.55%	83.46%	81.43%
Subject2	96.97%	96.87%	93.94%	93.36%	90.44%	88.23%	83.64%	78.72%	87.13%	83.96%	92.56%	92.10%
Subject3	93.09%	92.30%	91.38%	90.57%	90.81%	88.82%	89.52%	88.02%	88.79%	86.23%	88.33%	86.65%
Subject4	94.60%	93.99%	90.72%	89.26%	77.67%	74.36%	85.94%	83.70%	95.13%	94.75%	90.81%	89.99%
Subject5	96.78%	96.77%	96.97%	97.02%	88.14%	84.88%	88.14%	84.86%	82.81%	81.22%	84.93%	81.93%
Subject6	99.24%	99.23%	96.69%	96.57%	92.37%	91.77%	89.71%	87.38%	93.11%	91.59%	97.89%	97.90%
Subject7	94.51%	94.02%	96.40%	96.25%	88.60%	86.55%	91.45%	89.87%	94.49%	94.16%	86.76%	83.14%
Subject8	87.41%	85.54%	87.22%	85.26%	83.82%	80.74%	84.01%	80.72%	76.75%	72.48%	86.12%	84.84%
Overall	**94.28%**	**93.67%**	**93.12%**	**92.36%**	**87.61%**	**85.36%**	**88.03%**	**85.50%**	**88.53%**	**86.62%**	**88.86%**	**87.25%**

sensing modalities leads to a severe performance degradation. Once again, the overall performance of SADeepSense and DeepSense are very close to each other, mainly due to the utilized mean query in sensor attention of SADeepSense. Considering the diverse reliability of sensing modalities, using a mean query for attention weight estimation is obviously not an optimal solution, which is even worse than the brute-force attention design in BANet. The best position still belongs to our GlobalFusion framework. The GlobalFusion-Single model improves the back-bone DeepSense by 11.57% in accuracy, and the GlobalFusion model further improves GlobalFusion-Single by 0.78%. From the confusion matrix of GlobalFusion in Figure 5.4 (b), we can see that most classes are classified accurately, but there is still an ambiguity between sitting and standing classes. This observation is similar as our finding in PAMAP2, which also results from the dominant effect of accelerometer features. One more point we want to mention is that GlobalFusion-SGL has a slightly better macro F1 score than GlobalFusion, which means it's more stable across classes in classification.

DSADS Results: According to the classification results on DSADS in Table 5.4, attentional models generally work better than non-attentional DeepSense framework. It means all



(a) PAMAP2 (b) RealWorld-HAR



(c) DSADS

Figure 5.4: Normalized confusion matrix of GlobalFusion on PAMAP2, RealWorld-HAR and DSADS. (Every figure is the average overall all subjects.)

attention designs have a positive influence in model performance, so we are expecting to see an unbalanced attention weight distribution here (as validated by Figure 5.10), in contrast to the homogeneous information fusion in DeepSense. SADeepSense outperforms DeepSense only by 0.42%, due to its defective mean query design, while our GlobalFusion-Single and GlobalFusion consistently improve the model performance, achieving an accuracy of 93.12% and 94.28% respectively. The normalized confusion matrix of GlobalFusion is given in Fig-

Table 5.5: DG classification result.

Test User	GlobalFusion-SGL		DeepSense		SADeepSense		attnLSTM		BANet	
	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1	Acc.	Mac. F1
Subject1	92.71%	64.20%	89.40%	60.88%	86.16%	57.45%	90.63%	60.00%	92.41%	61.83%
Subject2	90.18%	71.10%	89.84%	65.07%	90.16%	66.74%	88.59%	60.04%	89.22%	56.52%
Subject3	86.15%	65.45%	86.46%	64.95%	86.56%	66.49%	84.48%	62.78%	85.94%	57.56%
Subject4	99.58%	49.90%	98.24%	49.56%	98.05%	49.51%	97.36%	49.33%	99.22%	49.80%
Subject5	83.65%	66.68%	80.42%	52.86%	81.98%	61.14%	81.56%	60.24%	81.56%	59.16%
Subject6	93.65%	56.57%	94.06%	51.75%	94.06%	55.92%	93.44%	48.30%	93.75%	48.39%
Subject7	96.22%	73.58%	95.18%	74.07%	93.36%	70.81%	92.32%	65.54%	94.40%	68.25%
Subject8	79.17%	72.96%	69.69%	66.33%	69.69%	64.23%	63.12%	61.16%	72.81%	61.58%
Subject9	88.15%	67.72%	85.94%	53.47%	86.42%	58.85%	86.78%	61.97%	85.94%	52.19%
Subject10	100%	100%	100%	100%	99.82%	49.95%	99.91%	49.98%	100%	100%
Overall	**90.94%**	**68.82%**	**88.92%**	**63.89%**	**88.62%**	**60.11%**	**87.82%**	**57.93%**	**89.52%**	**61.53%**

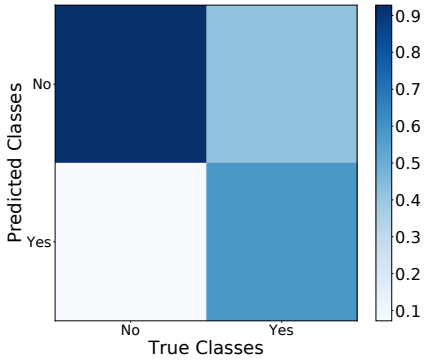


Figure 5.5: Normalized confusion matrix of GlobalAttention-Single on DG. (The figure is the average result overall all subjects.)

ure 5.4 (c). In this dataset, we can find the misclassification between sitting and lying right classes. We observe that most attention weights are still allocated to the accelerometer, which is not good at distinguishing between static gestures.

DG Results: At last, we analyze the classification results on DG. This is a binary classification problem where training data is distributed rather unbalanced between two classes. Most (over 95%) data samples belong to the 'Not freezing' class. We do not add any data augmentation techniques in data preprocessing, such as down-sampling or up-sampling of unbalanced classes. Since we only have accelerometer data from several data positions, only global position attention module is available here. As indicated in Table 5.5, the advantage of GLobalFusion-SGL is not obvious as on other three datasets. Only 1.42% improvement in accuracy is observed compared to the best baseline model, BANet. The heterogeneity among different spatial positions is not as large as diversity among different sensing modalities. Furthermore, none of the models can overcome the unbalanced class problem, since all of them show very low F1 score. Same conclusion can also be drew

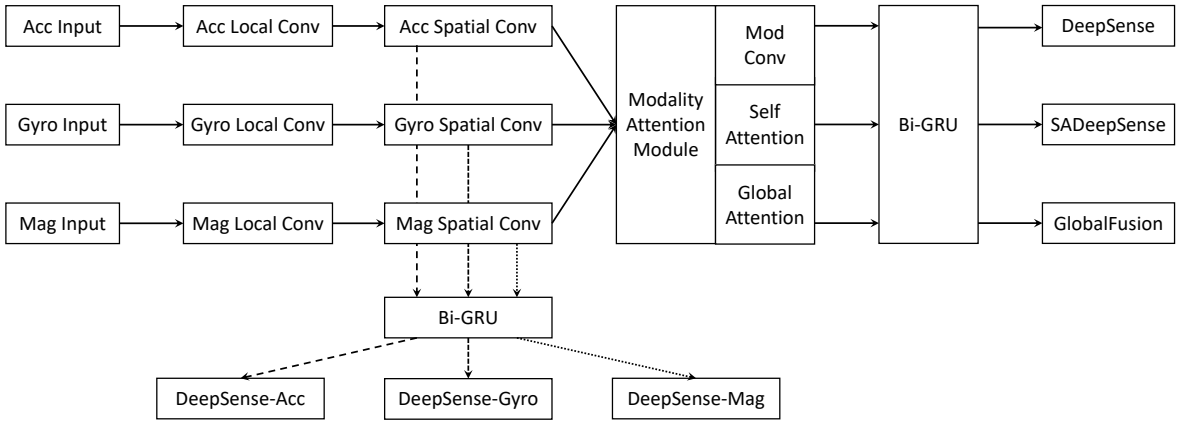


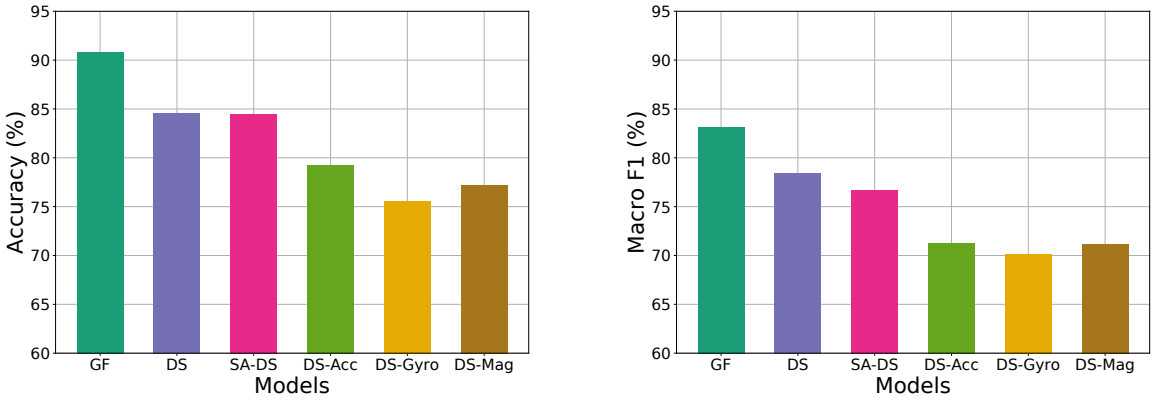
Figure 5.6: Information fusion capability evaluation logic example. (We use different dash lines for each individual sensor model.)

from the normalized confusion matrix of GlobalFusion in Figure 5.5. A large portion of positive samples are still misclassified as negative, which is especially unacceptable in medical applications considering the safety of patients. In summary, the usage of global attention module is not as beneficial as in other datasets, and it is not powerful enough to overcome the unbalanced class distribution problem. Appropriate data augmentation techniques are still needed to address this issue, while the positive aspect is that attention design is independent of data augmentation so they can be applied together.

5.4.5 Information Fusion Capability

In order to further explore the impact of global attention module in general classification performance, in this part, we look into details about information fusion capabilities between different attention mechanisms. All models are based on the back-bone design of DeepSense. To make the comparison fair and straightforward, we use the same lower level structures (*i.e.*, below modality convolution module) at each model. An example to illustrate the evaluation logic of this subsection is shown in Figure 5.6. We first show the prediction performance of each single-sensor model, and then compare the information fusion capability between different modality attention modules, including convolution operation (*i.e.*, DeepSense), self-attention (*i.e.*, SADeepSense), and our global attention (*i.e.*, GlobalFusion-Single). No position attention module is applied here. Evaluations are performed on three datasets: PAMAP2, RealWorld-HAR, and DSADS. All figures are results based on the k-fold cross-validation on all subjects in each dataset.

PAMAP2: The comparison result of PAMAP2 is presented in Figure 5.7. Accuracy and



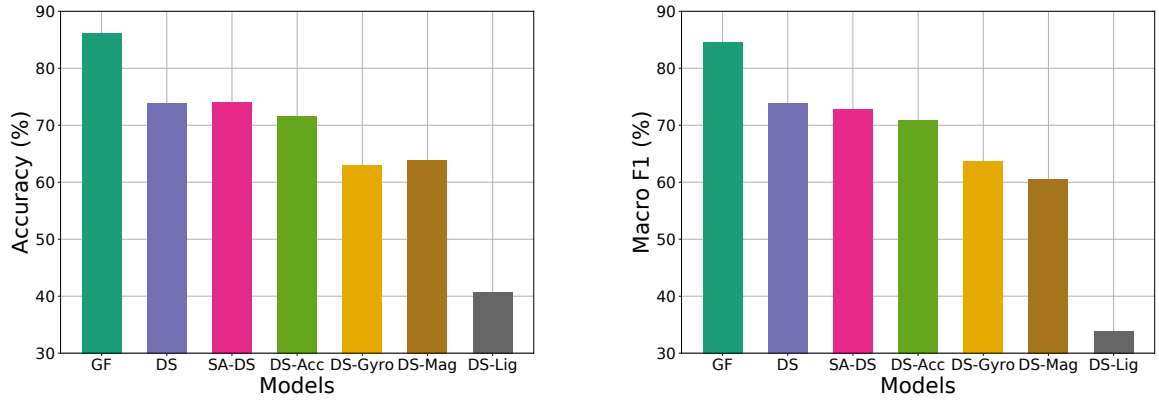
(a) Accuracy

(b) Macro F1 Score

Figure 5.7: Information fusion comparison on PAMAP2.

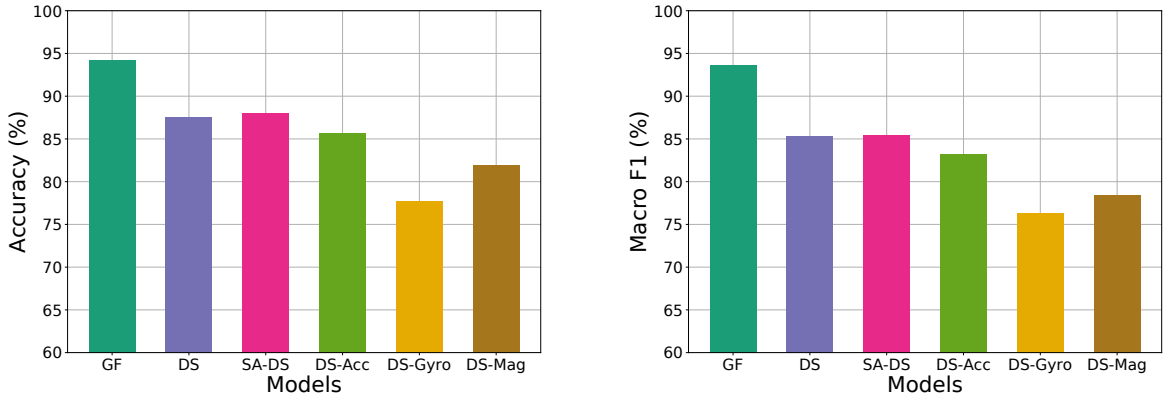
macro F1 score of each single-sensor model and composite model are shown. We can see that when three sensing modalities are used individually, accelerometer and magnetometer achieve relatively better performance than gyroscope. All of three modality fusion modules can take use of the information from three sensing modalities, because all of them can beat the best single sensor model (*i.e.*, DeepSense-Acc). Among the three referred attention modules, GlobalFusion has the best information fusion capability because it possesses the highest accuracy and macro F1, while the performance of SADeepSense and DeepSense are inferior to GlobalFusion but similar to each other. We also find that accelerometer data shows highest reliability in classification when used individually. Similar observations will also be given in other two datasets, and we will try to find the connection between single sensor performance and its corresponding attention weight returned by GlobalFusion in next subsection.

RealWorld-HAR: The comparison result of RealWorld-HAR is shown in Figure 5.8. Four sensing modalities achieve quite diverse performance, where the accuracy of DeepSense-Acc is clearly higher than DeepSense-Gyro and DeepSense-Mag, while the performance of DeepSense-Light is much worse than all other sensors. SADeepSense achieves similar performance as DeepSense, both of which are slightly better than DeepSense-Acc. It means that these two models is still able to extract complementary information from other sensing modality features excluding accelerometer feature. Our GlobalFusion beats these two models by a large margin, improving the performance of DeepSense by about 11%. Meanwhile, DeepSense-Acc shows an obviously better performance than other three single-sensor models. This observations is same as what we have saw in PAMAP2. We also dig into reasons behind the failure of light sensor model. After checking the raw data, we found that



(a) Accuracy (b) Macro F1 Score

Figure 5.8: Information fusion comparison on RealWorld-HAR.

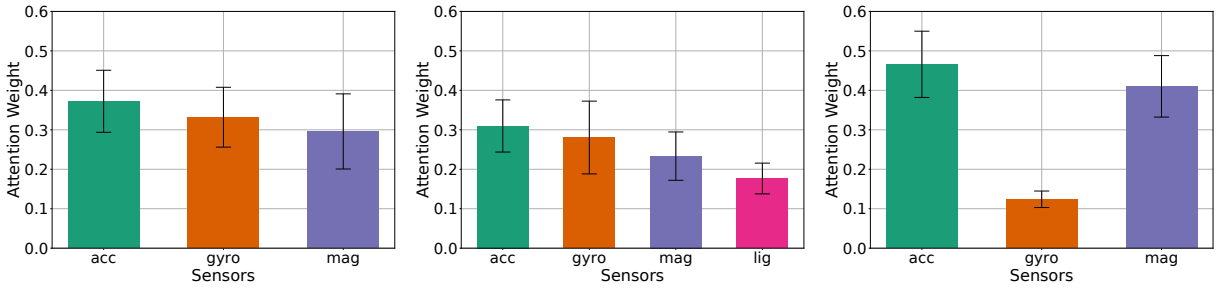


(a) Accuracy (b) Macro F1 Score

Figure 5.9: Information fusion comparison on DSADS.

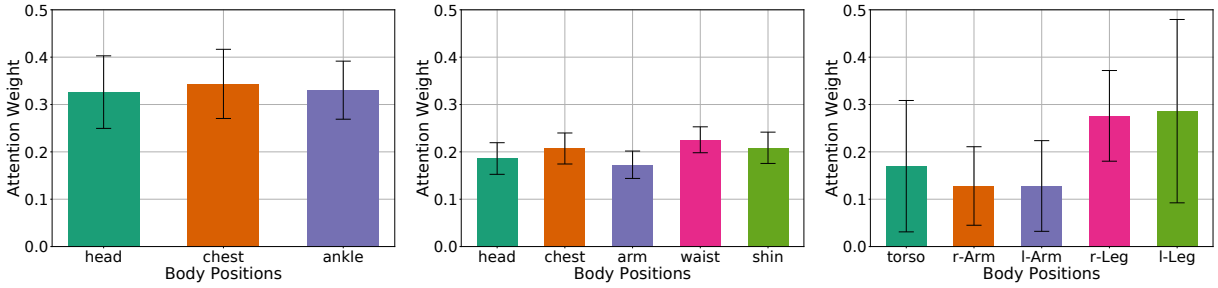
although the light sensor maintains a 50 Hz sampling rate, there is not much vibration in its readings. In most cases, there is no value change for light sensor within each time interval. Therefore, the information contained in light data is much lower than other sensor types.

DSADS: At last, we compare the information fusion result on DSADS. The corresponding accuracy and F1 results are given in Figure 5.9. As we have observed in all previous datasets, regarding the single sensor model performance, DeepSense-Acc > DeepSense-Mag > DeepSense-Gyro. This result will be integrated into the attention weight analysis in next part. For the sensor fusion models, conclusion are similar: DeepSense and SADeepSense share similar accuracy and macro-F1 score, because they both prefer more homogeneous input from different sensing modalities. Since GlobalFusion does not follow this assumption,



(a) PAMAP2 (b) RealWorld-HAR (c) DSADS

Figure 5.10: Modality attention weight distribution on different datasets.



(a) PAMAP2 (b) RealWorld-HAR (c) DSADS

Figure 5.11: Position attention weight distribution on different datasets.

its performance is much better than the other two. In summary, we have shown the superior information fusion capability of GlobalFusion compared to DeepSense and SADeepSense from the pure data driven aspect. To further validate the reasonability of our design, we directly look at and analyze the returned attention weight distribution, and try to interpret it from the physical aspect.

5.4.6 Qualitative Analysis on Attention Weight

In this subsection, we give a qualitative analysis about how our global attention design deals with the heterogeneity in information fusion, as well as some observations we get through analyzing the attention weight distribution. We will also answer the question about how we decide the information fusion order from the perspective of heterogeneity level.

First, we only use modality attention module in GlobalFusion, and analyze the statistical distribution of attention weight among each sensing modality. The results are given in Figure 5.10. The accelerometer always has the best classification result, and it also receives most attention by GlobalFusion on all datasets, which is what we have anticipated. However, on all three datasets, we observe a better performance on DeepSense-Mag than

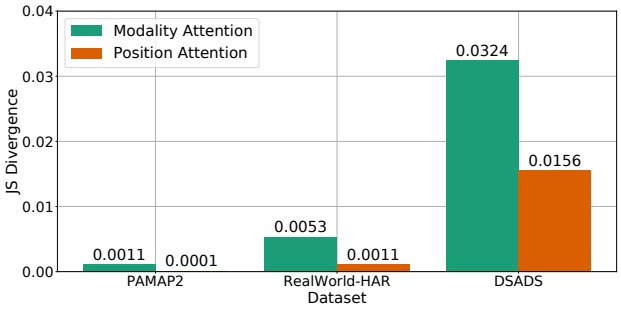


Figure 5.12: Attention weight divergence on each dataset

DeepSense-Gyro, but magnetometer has a higher attention weight than gyroscope in both PAMAP2 and RealWorld-HAR. We try to understand this observation from the physical principles: Accelerometer measures the absolute moving speed patterns, while magnetometer measures absolute orientation pattern in NESW plane. Gyroscope measures the changes in the orientation. If we take the derivative of consecutive magnetometer measures, we get the angular velocity in NESW plane; similarly, when we take the integral of consecutive gyroscope measures, we get the angular changes. Thus, gyroscope and magnetometer readings unavoidably contain overlapped information. At meantime, magnetometer lacks the vertical plane orientation information, while gyroscope lacks the absolute facing direction information of sensors to transferring the angular velocity from absolute earth coordinates to human body coordinates [142]. Therefore, the two sensors are also complementary to each other. The information contained by accelerometer readings is more independent of gyroscope and magnetometer, and is more critical in deciding human gesture patterns (i.e. activities), so it is supposed to receive highest attention weight. The information of gyroscope and magnetometer are both overlapped and complementary, so the relation between their attention weights is not fixed and can be affected by noise level of sensors. For the light sensor used in RealWorld-HAR, as we have mentioned before, although it has a completely different sensing principle compared to other inertial sensors, the lack of vibrations in its readings (at least in this dataset) restricts it to contain much information in frequency domain, so that it only gets a very low attention.

Next, we only include position attention in GlobalFusion, and present the results in Figure 5.11. Our observations are in two folds. First, the divergence of attention distribution over different body positions is smaller than different sensing modalities, which means that the information contained across sensing modalities are more diverse compared to spatial positions. To further validate this hypothesis, we show the JS divergence between both attention distributions and uniform distribution on each dataset in Figure 5.12. JS divergence

is defined based on KL divergence as follows:

$$JS(P \mid\mid Q) = \frac{1}{2}KL(P \mid\mid M) + \frac{1}{2}KL(Q \mid\mid M), \tag{5.5}$$

$$\text{where:} \qquad M = \frac{1}{2}(P + Q), \tag{5.6}$$

$$KL(P \mid\mid M) = \sum_{x} P(x) \log \left(\frac{P(x)}{M(x)} \right). \tag{5.7}$$

P and Q are two normalized distributions. We can see from Figure 5.12 that the JS divergence in modality attention distribution is apparently larger than position attention distribution on each dataset. This is exactly the reason why we choose to merge position information first before sensing modalities. We want to push the fusion of more heterogeneous information to higher layers of network so that we do not break the information integrity at lower level feature extraction. The second observation is, in both RealWorld-HAR and DSADS, arm sensors both get lower attention weight than other positions. Our interpretation is that although the moving patterns of arm contains a lot of information because its movement range is larger than other body positions, this information is not necessarily closely related to the target classes (*i.e.*, activities). Instead, the information contained in the arm movement can actually make confusion to the model. Therefore, arm features are not assigned large attention weight by the global attention module. By all the above observations and analysis, we prove that our global attention design is logically reasonable and agrees well with existing human knowledge.

5.4.7 Time and Energy Efficiency

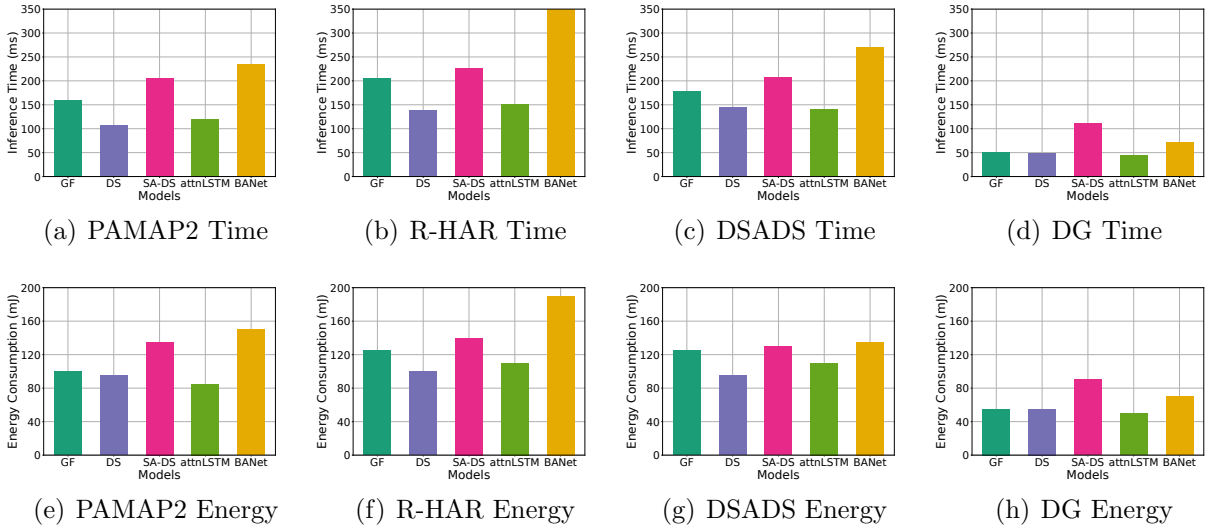
In this part, we evaluate the time and energy efficiencies of GlobalFusion when deployed on IoT devices. The experiments are conducted on two types of IoT devices, LG Nexus 5 and Raspberry Pi Model B. Nexus 5 is powered by a 2.26 GHz quad-core Snapdragon 800 processor with 2 GB of RAM, 32 GB of internal storage, and a 2300 mAh battery. The installed operating system is Android 7.1.1. Raspberry Pi 3 Model B is powered by a quad core 1.2 GHz Broadcom BCM2837 64bit CPU with 1 GB RAM and 16 GB storage. The preinstalled Raspbian Jessie operating system is used for Raspberry Pi. For all the models, we only use on-chip CPU for inference. Every model is preloaded to IoT device before experiment, and any unnecessary application and service that may interfere model computation are closed in advance. During the runtime on Nexus 5, we use the TensorFlow

Lite interpreter [167] as the inference engine, which is specially designed for running deep learning models on mobile, embedded, and IoT devices. Since TensorFlow Lite on Python is still under development during the paper writing, we use vanilla TensorFlow library for inference on Raspberry Pi. The energy consumption is measured by an external Monsoon High Voltage Power Monitor [168]. We independently run each model on each dataset for 20 times and take the average of their inference time and energy consumption.

The time and energy efficiency results of inference on Nexus 5 are shown in Figure 5.13, while the results on Raspberry Pi 3 Model B are shown in Figure 5.14. The results on both devices share the following common observations: First, although both GlobalFusion and SADeepSense are based on the backbone structure of DeepSense, GlobalFusion always leads to shorter inference time and less energy consumption, because we do not utilize the multi-head design in our global attention module. The additional time and energy overhead of GlobalFusion compared to DeepSense is within an acceptable range. Second, the attnLSTM is most time and energy efficient across all datasets, because it has less layers than all compared models. In addition, they first fuse information of different sensing modalities at each specific intervals (*i.e.*, which is same as our design) before merging information across intervals, so that only one recurrent module is used. Third, SADeepSense and BANet show the worst time and energy efficiency. SADeepSense has the most layers in its architecture and utilize multi-head design in both sensor and time attention modules. BANet first aggregates information across intervals at each individual sensor before fusing information across the sensing modalities. One individual recurrent unit is required by each sensor. Since the RNN computations are typically time consuming, we can expect to see a larger time difference between BANet and all other models when utilizing GPU for computations, because GPU is optimized for parallel computation of convolutional operations. The inefficiency of BANet is amplified especially we have more sensor types and body positions to fuse information (*i.e.*, BANet is slower than all other models on both RealWorld-HAR and DSADS with a large margin). Four, in most cases of our overhead evaluations, energy efficiency shares the similar results with time efficiency because we only use CPU for computations. A more complex relation between time and energy consumption will appear when GPU and DSP on the chip are involved in the computation, or cloud offloading is applied during the inference. This problem is beyond the scope of discussion in this work.

5.5 RELATED WORK

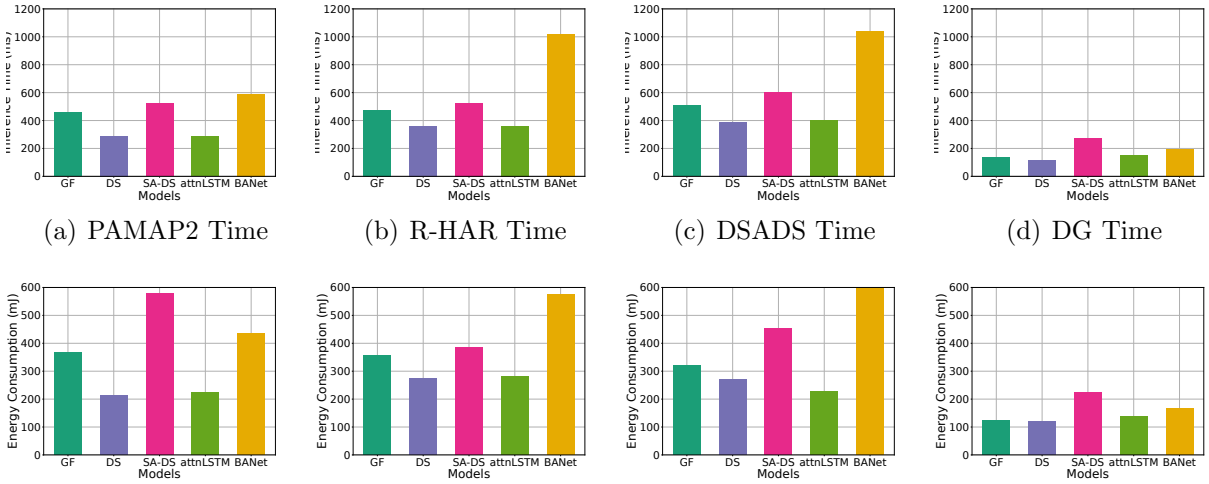
Textbooks have been written on classical multisensor data fusion [169] prior to the recent resurrection of neural networks research. These traditional techniques fuse carefully-designed



(a) PAMAP2 Time (b) R-HAR Time (c) DSADS Time (d) DG Time

(e) PAMAP2 Energy (f) R-HAR Energy (g) DSADS Energy (h) DG Energy

Figure 5.13: Time and energy efficiency on Nexus 5.



(a) PAMAP2 Time (b) R-HAR Time (c) DSADS Time (d) DG Time

(e) PAMAP2 Energy (f) R-HAR Energy (g) DSADS Energy (h) DG Energy

Figure 5.14: Time and energy efficiency on RaspBerry Pi 3 Model B.

features from each sensor thus calling for a human feature engineering effort. We shall not consider these approaches further as we seek a fully automated machine learning solution.

Deep learning revolutionized data fusion as it obviates feature engineering, instead ingesting raw sensor measurements only. Several deep-learning-based fusion methods have recently been proposed [147, 148, 170, 171, 172, 173]. For example, Yao *et al.* [148] concatenate different modality representations and use a convolution operation to fuse their information. Others explicitly model sensor interactions and correlations for better fusion quality [147, 170, 171, 172, 173]. While these approaches compute global context from local data, the "wiring" of global context as a function of local data is fixed. In contrast, an at-

tention mechanism allows selective retrieval of some local data to add to the global context for further processing, in essence allowing for different weighting of the same local feature in different global contexts.

Indeed, a key challenge in information fusion is to dynamically understand which inputs or features are more important when. This is akin to attention mechanisms in neural networks [16, 174, 175, 176]. It is an emerging technique for dynamically adjusting neural network model's focus by multiplying the features of each sensor with a corresponding weight, where the weight is dynamically estimated by an independent module based on the sensor inputs. Different solutions vary in their choice of weight calculation methods for fusion inputs [149, 159, 160, 161].

The attention mechanisms used in multisensor fusion are generally divided into two categories, called *additive attention* and *multiplicative attention*, respectively. In additive attention designs [160, 161], a small fully connected neural network is utilized to learn the weight of sensors/positions directly based on their inputs. In multiplicative attention, the weight of each sensor or position is decided by the compatibility between its features and a special feature vector, called *query*. The compatibility function is typically defined as a dot product of two vectors. In the sensor fusion problem, multiplicative attention is more intuitive because we can define the relevance of sensor features through a corresponding query design, unlike the black-box implementation of additive attention. The choice of the query directly decides the attention weight received by each sensor/position. For example, Yao *et al.* [159] use the mean of all sensor feature vectors as the query to estimate the attention weight of local features from each sensor component. They rely on the assumption that sensing information is highly correlated while the noise is not. However, this solution does not address different sensing modalities well because the information contained in fusion inputs are probably dissimilar but complementary to each other.

In contrast, in our design, we propose to use aggregated information from *higher layers* of the neural network to estimate the importance of local sensor features. We show that such an approach outperforms others because it is able to choose weights based on more advanced features, not available (*i.e.*, not yet computed) at lower layers of the neural network. A similar idea of using aggregated global information to selectively emphasize informative local features originated in recent efforts on applying convolutional neural networks (CNN) to image recognition [165, 177, 178, 179]. In Squeeze-and-Excitation (SE) block [177]. Hu *et al.* explore channel relationships (*i.e.*, RGB channels) in an image by stacking an information gathering stage (*i.e.*, squeeze block) with a following information distribution stage (i.e, excitation block) to adaptively recalibrate channel features. In [165], Jetley *et al.* leverage global image representation fed to the last classification layer as the query, to estimate

weights of local area features at intermediate convolution layers, after which the highlighted local features are output directly for classification. Both of them have observed significant improvement in image recognition accuracy.

To the best of our knowledge, we are the first to apply *global information based attention design* to *multisensor fusion* for IoT applications. Compared to the channels or local areas within an image, differences in the nature of information obtained from different sensors or spatial positions is a key challenge that hinders the direct application of global information based attention mechanism here. We tackle this challenge as follows. First, compared to [177], we explore a different method in information gathering stage. Instead of using a fully connected layer, we leverage the three-layer convolution module in DeepSese [148] to gather global information from sensors/spatial positions, which is known to be both effective and efficient in extracting representative features from multi-channel input. The gathered global information is used as the *global query* to recalibrate the *local features*. Second, compared to [165], we divide the information fusion into a hierarchy of two stages, named *modality fusion* and *position fusion*, respectively. Position fusion is performed before sensor fusion. The global information gathered from sensors/positions are immediately sent back to recalibrate the local features. We maximally preserve the flexibility in attention distributions to tackle information heterogeneity: Different sensors correspond to their own position attention distributions. Similarly, considering the time-varying sensing quality, the sensor fusion at different time intervals is independently performed with no information interference from other intervals.

CHAPTER 6: ATTENTION-BASED MISSING SENSOR RECOVERY

6.1 OVERVIEW

Internet of Things (IoT) advances promise great societal value at multiple scales, from single-device applications (*e.g.*, digital assistants [180, 181] and heart-rate monitoring [182, 183]), to multi-sensor applications (*e.g.*, autonomous driving [184, 185] and smart homes [186, 187]), and large-scale systems (*e.g.*, smart transportation [146, 188, 189, 190] and smart agriculture [191, 192]). In most applications, instead of relying on a single sensor, multiple embedded devices are interconnected to collaboratively carry out sensing tasks. Representative applications include protective behavior detection for people with chronic pain [160], patient health monitoring [193], traffic pattern analysis [188], and electroencephalogram (EEG)-based brain activity analysis [194]. This paradigm enables sensors to interact, collaborate, and learn from each other's observations. Deep neural networks [23, 39, 45, 48, 195, 196] have dominated learning and recognition tasks in such sensing-data-based applications, empowering an increasing spectrum of smart systems.

Reliable data delivery in IoT systems is crucial for providing high-quality services. We consider applications, where sensors from diverse locations are utilized together to perform intelligent spatially-distributed recognition tasks. We illustrate the processing pipeline of such systems in Figure 6.1. There are many reasons why data delivery from a sensor might be interrupted. They include battery depletion, power outages, weather conditions, communication failures, and external attacks. Human factors can also lead to missing sensor data. For instance, users might forget to wear a specific device that contributes data to the overall system. Therefore, the missing sensor problem is inevitable and frequently encountered when multi-sensor IoT systems are deployed in practice. In this work, we focus on situations where failures last for extended periods of time (compared to the sensor sampling period), so that missing sensor signals remain inaccessible before the issue is fixed.

Unfortunately, neural network models are generally sensitive to missing features, and lack the ability to automatically adjust themselves under these failures. There is very limited work in machine learning literature to investigate the impact of missing data. Most prior work is limited to considering *randomly* missing features out of a large feature set. [197] In this work, we consider the more challenging scenario, where only a partial sensor set is present during the inference phase, while the model is trained on the complete sensor set. We define this problem as a *missing sensor problem*. The scenario is challenging because features of missing sensors are lost in many consecutive sampling periods. Intuitively, this condition

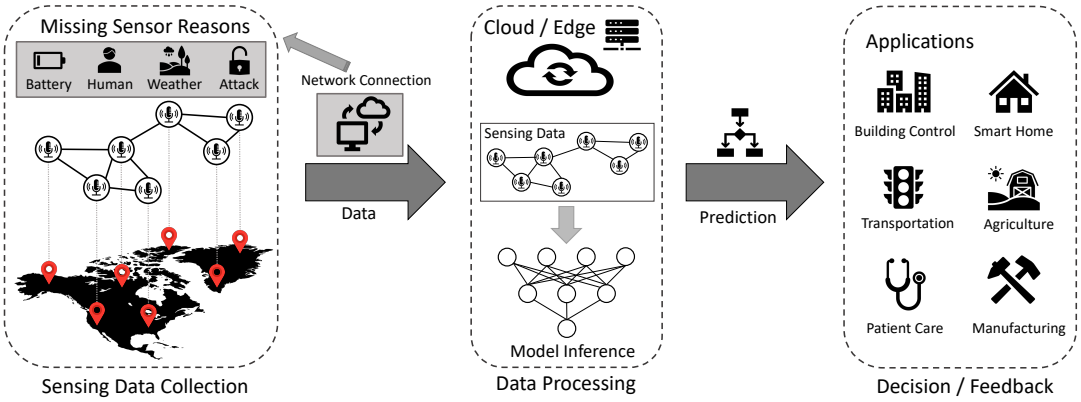


Figure 6.1: Overview of topology-aware IoT systems. Reasons that may lead to missing sensors are annotated in shadow area.

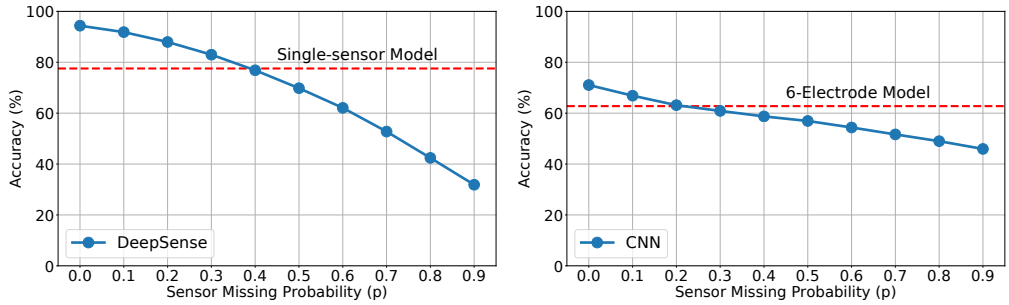
has a larger impact on model performance compared to randomly missing features, because most statistical imputation methods can no longer be used. We conducted an empirical experiment to examine the performance degradation caused by missing sensors on two IoT tasks; human activity recognition (HAR) using the DeepSense [148] framework and EEG-based motor-imagery recognition using a CNN framework proposed by Qiao *et al.* [198]. More details will be mentioned in Section 6.2. We saw an accuracy drop of 25% in the HAR task and 14% in the EEG task when input signals of 50% of the sensors are missing. Therefore, special attention needs to be paid to missing sensors in neural network inference.

A naive approach to dealing with missing sensors is to learn a separate model for every possible remaining sensor combination. By doing so, we can always maximally utilize the information contained in available sensors to achieve the best inference performance. However, this strategy will result in an exponential number of models, compared to the number of sensor nodes, which is unacceptable, especially when the number of sensors is large. For example, the EEG system usually contains 64 or 128 electrodes, while a smart transportation system can have hundreds to thousands of sensors deployed across the city. Therefore, most existing approaches have focused on training a single model that can work with different available sensor subsets. Vaizman *et al.* [199] randomly remove some sensors using dropout [200] at the input layer to make the neural network automatically learn to adapt to missing sensor data. However, the dropout strategy completely gives up the features at the missing sensor positions, which may cause more significant degradation. Instead, we try to recover the features of missing sensors by exploiting their spatial correlations with available sensors, to enhance behavior of the classification model. Some recent work [201, 202, 203] used a denoising autoencoder [204] to recover missing features. During training, they manually add random noise and turn off parts of the features of input data to make the autoencoder learn

to remove noise and recover missing features. It generally works well for randomly missing features, but is not able to handle missing sensors over extended periods.

In this work, we propose a novel feature reconstruction module, named the *graph recovery module*, for handling missing sensors in topology-aware IoT applications. We assume that there is a physical network, where the sensor nodes are deployed and connected. For example, in human activity recognition (HAR), the human skeleton structure can serve as a physical network for sensor/joint connections. Our insight is that sensing signals of physically adjacent sensor nodes are also physically correlated. During the reconstruction, the information is transmitted from available sensors to missing sensors to reconstruct their features. We design a novel neural message passing mechanism based on a graph convolutional network (GCN) [205]. However, information flow in original GCNs is not controlled, so the available sensor features can be polluted and missing sensor features are not perfectly reconstructed. To solve this problem, we further design a gating mechanism based on Gated Recurrent Units (GRUs) [206] to let the network learn to control the information flow between available sensors and missing sensors. Graph recovery is a flexible module that can be easily integrated into the state-of-the-art learning frameworks for IoT applications at different layers. In our implementation, we heuristically put it between individual sensor feature extraction module and sensor fusion module. The holistic network is trained in an end-to-end manner to make sure the reconstruction module and backbone network fit well with each other. Finally, to improve the balance of model performance under different missing sensor situations, we leverage a *hybrid training* strategy during model training. Multiple ratios of missing sensors are simultaneously emulated during training to guarantee the model works well under all missing sensor situations.

We choose two representative topology-aware IoT applications with corresponding state-of-the-art neural network frameworks to evaluate the reconstruction performance of the graph recovery module. They are human activity recognition (HAR) task trained with DeepSense [148] on two public datasets (REALDISP [207], RealWorld-HAR [156]), and EEG-based motor-imagery recognition trained with CNN [198] on the EEG-MMID [208] dataset. We evaluate the model performance as the missed sensor ratio varies from 0% to 90%. On one hand, the network equipped with our graph recovery module can achieve comparable performance to the original backbone network (with at most 1% accuracy loss), when all sensors are available. On the other hand, it can effectively maintain the model performance when more sensors are missing. There is only a 7% to 18% accuracy loss when 90% of the sensors are missing. In contrast, the accuracy of compared baseline algorithms drops by as much as 15% to 47% under the same conditions. Moreover, we successfully demonstrate the superiority of the graph recovery module in feature reconstruction by evaluating their



(a) Human Activity Recognition with DeepSense

(b) EEG-based motor-imagery Recognition with CNN

Figure 6.2: Accuracy of backbone networks when encountered with missing sensors. We use red dotted lines to show the accuracy of baseline models trained on ∼10% sensors.

reconstruction error against original features.

6.2 MOTIVATION

In this section, we investigate the impact of missing sensors on the performance of existing IoT recognition models. Previous work [197, 209, 210] empirically studied the impact of randomly missing features on neural network performance. When they feed samples with randomly missing features into a neural network, trained on complete features, a dramatic drop in model accuracy is observed. In our scenario, the features corresponding to missing sensors are missing consecutively over a long period of time. Experiments are performed on two multi-sensor applications, human activity recognition (HAR) and EEG-based motor-imagery recognition.

For human activity recognition (HAR), we train a DeepSense [148] model on the REALDISP dataset [207, 211], where 9 IMU sensors are deployed on the body. The model is trained on data from all sensors. During the evaluation, each sensor is removed independently with a probability p. We evaluate the model accuracy when p increases from 0 to 0.9. At least one sensor is present in each testing sample. The results are shown in Figure 6.2(a). In addition, we train a baseline model on single-sensor input. Its result is shown in red dotted line. We can see that the accuracy of DeepSense drops significantly when the missed sensor ratio increases. The baseline model accuracy reflects the amount of information contained in single-sensor samples, given an appropriately trained model, tailored for that specific sensor. It upper-bounds the performance of algorithms that do not train a separate model for each possible sensor combination. The large performance gap between the tailored single-sensor model and DeepSense ($p = 0.9$) indicates that DeepSense can not effectively utilize the information contained in available sensors when enough sensors are missing. It leaves plenty

of space for improvement.

Another experiment does electroencephalogram (EEG)-based motor-imagery recognition. In EEG test, a set of electrodes are placed along the scalp to measure voltage fluctuations at different points resulting from brain activity. We train a CNN model proposed by Qiao *et al.* [198] on the EEG-MMID [208] dataset. We use 64 electrodes, distributed according to the international 10-10 system [212]. The CNN model is trained based on data from all 64 electrodes, while each electrode is randomly dropped with probability p during inference. The accuracy curve is given in Figure 6.2(b), when p increases from 0 to 0.9. Besides, a baseline model trained on data from 6 randomly selected electrodes is shown in the red dashed line. There is also a clear performance drop observed along with the increasing sensor miss ratio p. Note that, the expected number of available electrodes is larger than 6 even when $p = 0.9$. However, the performance of the CNN model is worse than the baseline model for $p \geq 0.3$. We conclude that the CNN model is sensitive to missing sensors in EEG classification as well.

The neural networks used in the above two IoT applications show sensitivity when encountering missing sensors. To fill the performance gap and enhance model robustness, rather than training a separate model for each possible available sensor subset, a more practical approach is to design a dedicated module to reconstruct the features of missing sensors based on information contained in available sensors. This motivates developing our *graph recovery module*. We will introduce the related background, technical design details, and training strategy in the following sections.

6.3 PRELIMINARIES

In this section, we give a preliminary overview of Graph Neural Networks (GNNs) [205, 213], denoising AutoEncoders [204], and Gated Recurrent Units (GRUs) [206], which all lay a theoretical foundation for our design of the graph recovery module. Since we mainly use Graph Convolutional Networks (GCNs) in this work, we will only introduce GCN-related theories here.

Notations are defined as follows. All vectors are denoted by bold lower-case letters (*e.g.*, $\mathbf{x}$ and $\mathbf{y}$), while matrices and tensors are represented by bold upper-case letters (*e.g.*, $\mathbf{X}$ and $\mathbf{Y}$). For a vector $\mathbf{x}$, the j^{th} element is denoted by x_j. For a tensor $\mathbf{X}$, the t^{th} matrix along the first axis is denoted by $X_{t..}$, and other slicing denotations are defined similarly. For each layer k, we use $\mathbf{X}^{(k)}$ to denote the input to this layer, and use $\mathbf{Y}^{(k)}$ to denote corresponding output. For any tensor $\mathbf{X}$, $|\mathbf{X}|$ denotes the size of $\mathbf{X}$. All sets are denoted by upper-case calligraphic letters (*e.g.*, $\mathcal{V}$ and $\mathcal{E}$). Similarly, $|\mathcal{V}|$ denotes the size of $\mathcal{V}$.

6.3.1 Graph Convolutional Networks (GCNs)

Graph Convolutional Networks (GCNs) are a generalization of well-established Convolutional Neural Networks (CNNs) to non-Euclidean graph-structured data, such as social networks, the World Wide Web, traffic networks, and abstract knowledge graphs. It can leverage graph topology to aggregate node information from the neighborhood in a convolutional fashion. [213]. We focus on the widely-adopted GCN version proposed by Kipf and Welling in [205], which is a spectral-based graph convolution design with spatial localization meaning. Assume we have a graph $\mathcal{G}$ with N nodes, whose topology is represented by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. Its corresponding graph Laplacian is defined as,

$$\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top, \tag{6.1}$$

where $\mathbf{D} \in \mathbb{R}^{N \times N}$ is the diagonal degree matrix with $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$, $\mathbf{I}_N \in \mathbb{R}^{N \times N}$ is the identity matrix, $\mathbf{U}$ and $\mathbf{\Lambda}$ are eigenvectors and diagonal matrix of eigenvalues corresponding to $\mathbf{L}$. The spectral convolution on the graph is:

$$g_\theta \star \mathbf{x} = \mathbf{U} g_\theta \mathbf{U}^\top \mathbf{x}, \tag{6.2}$$

where $\mathbf{x} \in \mathbb{R}^N$ is a graph feature vector, and g_θ is the convolution kernel. The intuitive explanation is that we first do a graph Fourier transform on graph features by $\mathbf{U}^\top \mathbf{x}$, multiply it by the convolution kernel g_θ, and finally perform a reverse Fourier transform by multiplying it with $\mathbf{U}$. Next, we regard the convolution kernel as a polynomial function $g_\theta(\mathbf{\Lambda})$ of the diagonal eigenvalue matrix $\mathbf{\Lambda}$, so that the convolution becomes,

$$g_\theta \star \mathbf{x} = \mathbf{U} g_\theta(\mathbf{\Lambda}) \mathbf{U}^\top \mathbf{x} = \mathbf{U} \left(\sum_{k=0}^{K} \theta_k \mathbf{\Lambda} \right) \mathbf{U}^\top \mathbf{x} = \sum_{k=0}^{K} \theta_k \mathbf{L}^k \mathbf{x}, \tag{6.3}$$

where K is the chosen order of polynomial approximation, and θ are trainable parameters. To further improve the computational efficiency, we approximate $g_\theta(\mathbf{\Lambda})$ by its Chebyshev polynomials and set the order of approximation $K = 1$. The Chebyshev polynomials are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. To meet the requirement of Chebyshev polynomials, we normalize the eigenvalues as $\tilde{\mathbf{\Lambda}} = \frac{2}{\lambda_{max}}\mathbf{\Lambda} - \mathbf{I}_N$ to make them lie within $[-1, 1]$. λ_{max} denotes the largest eigenvalue of $\mathbf{L}$, which is assumed to be 2. After a few derivation steps, the graph convolution becomes:

$$g_\theta \star \mathbf{x} \approx \theta'_0 \mathbf{x} + \theta'_1 (\mathbf{L} - \mathbf{I_N}) \mathbf{x} = \theta \left(\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \right) \mathbf{x} = \theta \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{x}, \tag{6.4}$$

with a single parameter $\theta = \theta'_0 = -\theta'_1$, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ and $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. We finally generalize the definition to signal $\mathbf{X} \in \mathbb{R}^{N \times C_I}$ with C_I input channels and C_O output channels: $\mathbf{Y} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{\Theta}\right)$, where $\mathbf{\Theta} \in \mathbb{R}^{C_I \times C_O}$ is the trainable parameter matrix and σ is the sigmoid activation function. Though derived from the spectral domain, the graph convolution above is considered to have a clear meaning of spatial localization [213]. It is essentially equivalent to aggregating node representations from their direct neighborhood each time. As far as we know, we are the first to utilize GCNs from a message passing perspective to rebuild features of missing sensors based on the network's spatial topology structure.

6.3.2 The Denoising Autoencoder (DAE)

An autoencoder [214] is a neural network structure that is used to learn a useful encoding of data in an unsupervised manner. The output of an autoencoder is set the same as its input. In other words, it is designed to learn an identity function. It typically consists of an encoder that can transform the input data into a compressed latent representation, and a decoder that can recover the original input from the encoding of data. The learning objective of an autoencoder is to minimize the reconstruction loss between the decoder output and the original data,

$$\theta, \phi = \arg\min_{\theta,\phi} \|\mathbf{X} - D_\phi[E_\theta(\mathbf{X})]\|^2, \tag{6.5}$$

where E and D denote the encoder function and decoder function, respectively. A denoising autoencoder [204] is a variant of autoencoders that is specifically designed for data reconstruction from partially corrupted input. During the training, they randomly remove parts of the input and add random noise instead, before feeding it into the encoder. DAEs are trained to recover the original undistorted input. Its mathematical formulation is,

$$\theta, \phi = \arg\min_{\theta,\phi} \left\|\mathbf{X} - D_\phi[E_\theta(\tilde{\mathbf{X}})]\right\|^2, \tag{6.6}$$

where $\tilde{\mathbf{X}}$ denotes the distorted input data. Although we don't use the specific encoder-decoder structure in our graph recovery module, this training philosophy indeed motivates us to change the training of GCN to serve the purpose of filling missing sensor features.

6.3.3 Gated Recurrent Units (GRU)

Gated recurrent units (GRU) are a gating mechanism proposed by Cho *et al.* [206] to control the information flow in recurrent neural networks (RNN). It was originally introduced

to solve the vanishing gradient problem in standard RNN, especially when dealing with long sequences. GRUs use the so-called update gate and reset gate to decide what information should be passed to the output. Their advantage is that they can be trained to keep around older information without washing it through time, as well as removing information which is irrelevant to the prediction. Its mathematical formulation is as follows,

$$
\begin{aligned}
G_u &= \sigma\left(\mathbf{W}_u\left[\mathbf{h}_{t-1}, x_t\right] + b_u\right) \text{ [update gate]}, \\
G_r &= \sigma\left(\mathbf{W}_r\left[\mathbf{h}_{t-1}, x_t\right] + b_r\right) \text{ [reset gate]}, \\
\tilde{\mathbf{h}}_t &= \tanh\left(\mathbf{W}_m\left[\mathbf{x}_t, G_r \odot \mathbf{h}_{t-1}\right] + \mathbf{b}\right) \text{ [current memory]}, \\
\mathbf{h}_t &= G_u \odot \tilde{\mathbf{h}}_t + (1 - G_u) \odot \mathbf{h}_{t-1} \text{ [final memory]},
\end{aligned} \tag{6.7}
$$

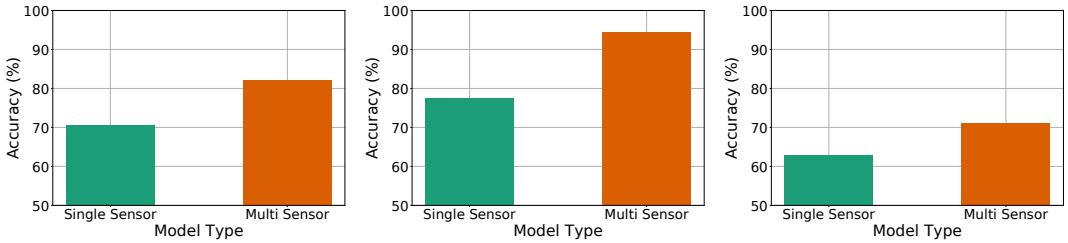
where h_t and h_{t-1} denote the latent representation of the current time step and previous time step respectively, x_t is the input to current time step, σ is the sigmoid function, and $\odot$ denotes the Hadamard (element-wise) product. The update gate helps the model to determine how much of past information needs to be passed along to the future, while the reset gate is used to decide how much of past information to forget. The final memory equation decides what to collect from current memory content $\tilde{\mathbf{h}}_\mathbf{t}$, and what from the previous steps $\mathbf{h}_{t-1}$. A similar idea can be used to control the information flow in the GCN-based message passing mechanism. At each message passing step, we need to find a tradeoff for each sensor between the information contained in this sensor, and messages passed from its neighboring sensors.

6.4 GRAPH-BASED MISSING SENSOR RECONSTRUCTION

With the previous foundation, we are ready to introduce our *graph recovery module* for reconstructing missing sensor features in IoT learning frameworks, in this section. We divide the description into three parts. We first give an overview of the general framework equipped with the graph recovery module. Next, we introduce the technical details of the graph recovery module. Finally, we discuss the design of loss functions and the training method.

6.4.1 Learning Framework with Feature Reconstruction

We consider applications where a set of sensors collaboratively capture an overall perception of the environment. Classification is performed at the graph level based on the joint information, as opposed to being performed individually at each sensor position. This is different from most application scenarios of graph neural networks (GNNs), but such applications are prevalent in the IoT field. For example, in the electroencephalogram (EEG)



(a) RealWorld-HAR [156] (b) REALDISP [207] (c) EEG-MMID [208]

Figure 6.3: Accuracy comparison between single sensor model and multi sensor model on HAR task (*i.e.*, RealWorld-HAR and REALDISP) and EEG task (*i.e.*, EEG-MMID). In EEG-MMID, we use 6 electrodes out of 64 as the "single sensor" model, which is about 10% of all available sensors.

test [215], a set of electrodes are placed along the human scalp to measure voltage fluctuations resulting from ionic current within neurons. In elderly/patient monitoring systems, a set of IMU sensors are placed on different body positions to recognize their activity.

Compared to single-sensor models, collaborations among sensors can significantly improve the accuracy and robustness of recognition. We empirically prove it in Figure 6.3. We compare the performance of a multi-sensor model and a single-sensor model on both HAR task and EEG task with three datasets. The multi-sensor model utilizes data from all sensor positions, while the single-sensor model only uses one or 10% out of available sensors. We can see in both tasks that, when all sensors are present, the multi-sensor model has a significantly higher accuracy than the single-sensor model. We seek to improve the robustness of the multi-sensor neural network models against missing sensors in IoT scenarios.

In a multi-sensor recognition task, we can generally partition the neural network into three stages: sensor feature extraction, sensor fusion, and classification. We use the HAR task to help understand the pipeline. In the sensor feature extraction stage, we try to learn the time/frequency patterns from each local position (*e.g.*, the hand movement patterns or the waist movement patterns). Then, in the sensor fusion stage, we aggregate all pieces of local information to recover the global patterns (*e.g.*, the whole body movement patterns). The extracted global features are then fed into the classification module for final inference. In HAR, activity recognition is based on the whole body movement patterns, instead of the local hand movement patterns.

To tackle the challenge caused by missing sensors, it is not enough to simply emulate all possible situations during the training, because the features of complete samples and partial samples are not fully compatible with each other. For example, in HAR, the complete features contain the whole body movement patterns, but the partial features may only contain upper body movement patterns. We have to explicitly teach the neural network to

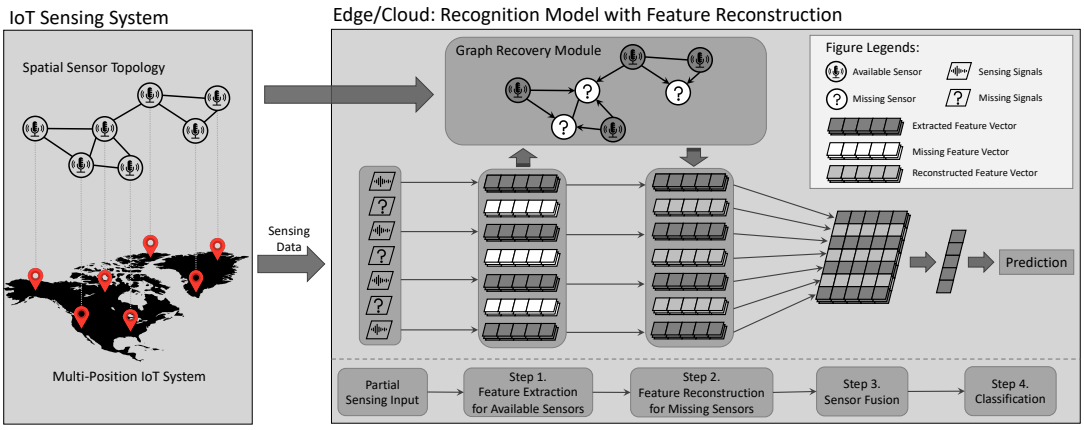


Figure 6.4: Overview of IoT deep learning framework equipped with graph recovery module. It consists of four steps: 1) Extract local features for available sensors; 2) Reconstruct features for missing sensors with the graph recovery module; 3) Perform information fusion on all sensors; 4) Feed aggregated features to classification model to get the prediction output.

use partial body features to recover the whole body patterns before feeding them into the classification module, so that the classification module is able to always base on the whole body patterns to perform prediction.

We heuristically put the missing sensor reconstruction between the sensor feature extraction stage and sensor fusion stage. An overview of our general IoT deep-learning framework, equipped with the feature reconstruction module, is given in Figure 6.4. In next part, we motivate use of spatial topology to do missing sensor recovery and explain how we modify Graph Convolutional Networks (GCNs) to serve this purpose.

6.4.2 Graph Recovery Module Design

In the previous section, we explained why we need a particular missing sensor reconstruction module and where to deploy this module. Next, we illustrate why and how we implement this functionality based on graph convolutional networks (GCNs).

The primary innovation in our design is that we try to integrate the physical spatial topology of sensors into the feature reconstruction module within the neural network. This idea is partially motivated by conventional data imputation methods widely adopted in spatial-temporal sensing data [216, 217], where the locality in time and space is exploited to fill randomly missing sensor readings. However, it has been proved that such methods, like Non-negative Matrix Factorization (NMF) [218], can't handle cases where the sensors are completely missing all the time [216], because they impute each single value independently so that the assembled sequence can not preserve the semantic meaning contained in the original

sensing signals. In the latent feature space, there is no semantic meaning contained in each numerical value, but information is contained in the high-dimensional vector as a whole. In our design, we reconstruct the feature vector of missing sensors as a basic unit. Meanwhile, unlike previous autoencoder based feature reconstruction methods [201, 202, 203] that exploit sensor correlations in a fully-connected manner, we only explicitly exploit sensor correlations in the latent space based on their spatial topology. On one hand, we can avoid information flooding at missing sensors where too much noise might overwhelm useful information; on the other hand, we can improve computational efficiency to a large extent, because the number of edges in most real-world networks tends to grow linearly (not quadratically) with the number of nodes [219]. Next, we elaborate how to interpret GCN as a special message passing mechanism, and how to control information flow within GCNs by a gating mechanism.

Understanding GCN as a Neural Message Passing Mechanism: As we mentioned before, one interesting property of the GCN proposed by Kipf and Welling [205] is that it has a clear meaning of vertex localization that bridges the gap between spectral-based methods and spatial-based methods. In this part, we describe how to understand it from a neural message passing perspective, and how the correlations between sensors are exploited during the training.

We are given a sensor topology graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = N$ nodes, and an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. Assume that we only know the connections between nodes, but not the weights on the edges, which means $\mathbf{A}_{ij} = 1$ if there is an edge between the two sensors (*i.e.*, $(i, j) \in \mathcal{E}$); otherwise, $\mathbf{A}_{ij} = 0$. We use $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N] \in \mathbf{R}^{N \times C}$ to denote the feature matrix we need to recover, where C is the dimensionality of the feature vector on each sensor. Let's divide the sensors into two sets, the available sensor set $\mathcal{A}$ and the missing sensor set $\mathcal{M}$, where $\mathcal{A} \cup \mathcal{M} = \mathcal{V}$ and $\mathcal{A} \cap \mathcal{M} = \emptyset$. Besides, we use $\tilde{\mathbf{H}} = [\tilde{\mathbf{h}}_1, \tilde{\mathbf{h}}_2, \ldots, \tilde{\mathbf{h}}_N] \in \mathbf{R}^{N \times C}$ to represent the distorted sensor features, where the features of missing sensors are set to 0,

$$\tilde{\mathbf{h}}_i = \begin{cases} [0, 0, \ldots, 0], & \text{if } i \in \mathcal{M}; \\ \mathbf{h}_i, & \text{if } i \in \mathcal{A}. \end{cases} \tag{6.8}$$

$\tilde{\mathbf{H}}$ is the input to the graph recovery module. The objective of the *graph recovery module* is to reconstruct features of all missing sensors $i \in \mathcal{M}$. We follow the general neural message passing definition raised by Gilmer *et al.* [220]. We use $\mathbf{h}_i^t$ to denote the sensor i's hidden representation after t rounds of message passing, so we should have $\mathbf{h}_i^0 = \tilde{\mathbf{h}}_i$. A typical neural message passing mechanism consists of two components: message function M_t, and vertex update function U_t. The message function is the aggregated information sent from

neighbors:

$$\mathbf{m}_i^{t+1} = \sum_{j \in \mathcal{N}_i} M_{t+1}(\mathbf{h}_i^t, \mathbf{h}_j^t), \tag{6.9}$$

where $\mathcal{N}_i$ is the set of i's neighbor sensors. The passed message can depend on information available at both ends, as well as the edge information if available. The vertex update function is used to update the sensor's latent representation based on node representation from the previous step and the passed message at the current step,

$$\mathbf{h}_i^{t+1} = U_{t+1}(\mathbf{h}_i^t, \mathbf{m}_i^{t+1}). \tag{6.10}$$

Specifically in our case, according to the GCN definition: $\mathbf{H}^{t+1} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^t\mathbf{\Theta}^t\right)$, the node is updated as,

$$\mathbf{h}_i^{t+1} = \sigma\left(\tilde{\mathbf{D}}_{ii}^{-\frac{1}{2}}\tilde{\mathbf{A}}_{i\cdot}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^t\mathbf{\Theta}\right) = \sigma\left(\left(\mathbf{\Theta}^{t+1}\right)^\top \sum_{j \in \mathcal{N}_i}\left(\tilde{\mathbf{D}}_{ii}\tilde{\mathbf{D}}_{jj}\right)^{-\frac{1}{2}}\tilde{\mathbf{A}}_{ij}\mathbf{h}_j^t\right), \tag{6.11}$$

where $\mathbf{\Theta} \in \mathbb{R}^{C_t \times C_{t+1}}$ is a dimension mapping parameter matrix. To solve the issue that we only know the sensor connectivity but not their connection weights, we multiply an extra learnable adjacency weight matrix $\mathbf{W}_A$ by the adjacency matrix. It is shared across every message passing step/layer in the graph recovery module. Correspondingly, we have: $\tilde{\mathbf{A}}' = \mathbf{W}_A(\mathbf{A} + \mathbf{I_N})$ and $\tilde{\mathbf{D}}'_{ii} = \sum_j \tilde{\mathbf{A}}'_{ij}$. Possible weight sharing methods can be used in $\mathbf{W}_A$ if more domain knowledge is available. In other words, we use this adjacency weight matrix to describe sensor correlations in the latent space. Now, the message passing function and vertex update function are,

$$M_{t+1}(\mathbf{h}_i^t, \mathbf{h}_j^t) = \left(\tilde{\mathbf{D}}'_{ii}\tilde{\mathbf{D}}'_{jj}\right)^{-\frac{1}{2}}\tilde{\mathbf{A}}'_{ij}\mathbf{h}_j^t, \tag{6.12}$$

$$U_{t+1}(\mathbf{h}_i^t, \mathbf{m}_i^{t+1}) = \sigma\left(\left(\mathbf{\Theta}^{t+1}\right)^\top \mathbf{m}_i^{t+1}\right). \tag{6.13}$$

However, in this implementation, the difference between available sensors and missing sensors is not considered. Intuitively, when the available sensors are reconstructing missing sensors, the missing sensors are also polluting available sensors with their null messages. We will solve this problem in next part by replacing the vertex update function.

Information Flow Control in GCN-based Message Passing: One problem in the previously introduced GCN-based message passing mechanism is that there is no control on the information flow between available sensors and missing sensors. We solve this problem in three ways: message rescaling, a gated update function, and output substitution. We use

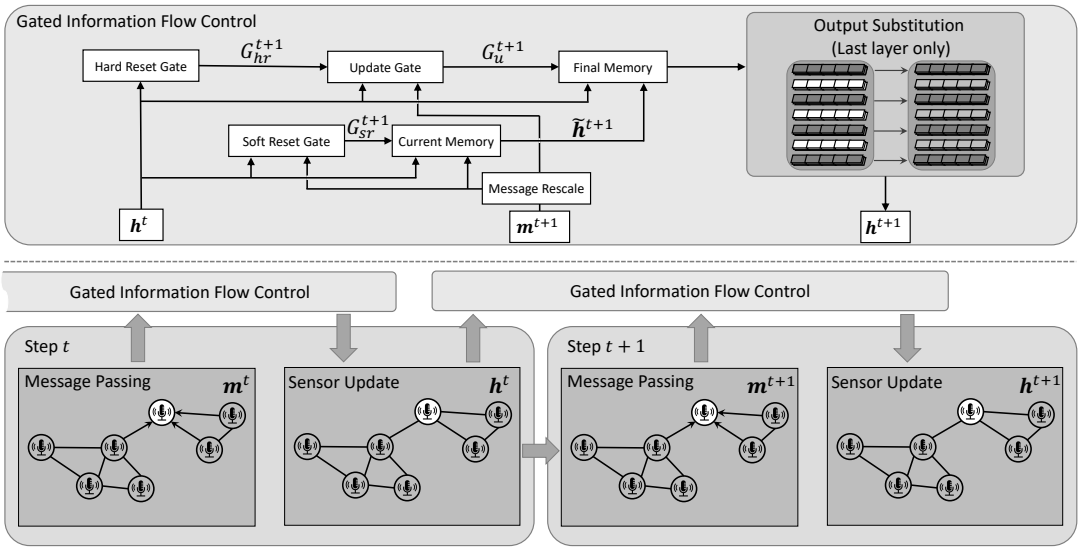


Figure 6.5: Message passing with controlled information flow diagram of the graph recovery module. We only show one layer/step of message passing. The graph recovery module is a stack of T layers of such an implementation. At the upper part, we show design of gated information flow control; at the lower part, an integrated message passing mechanism is presented.

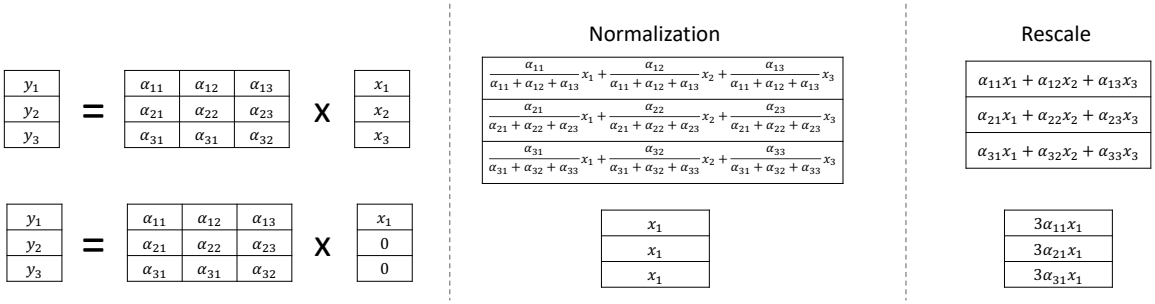


Figure 6.6: A toy example to explain why we choose message rescale, instead of normalization.

a diagram in Figure 6.5 to illustrate information flow control in graph recovery layers.

Message Rescaling. In vanilla GCNs, despite the dimension mapping, sensor updates purely take a certain weighted average of its neighboring nodes at each step. When considering the missing sensors, the received message at each node is,

$$\mathbf{m}_i^{t+1} = \sum_{j \in \mathcal{N}_i, \|\mathbf{h}_j^t\|>0} \left(\tilde{\mathbf{D}}'_{ii}\tilde{\mathbf{D}}'_{jj}\right)^{-\frac{1}{2}} \tilde{\mathbf{A}}'_{ij}\mathbf{h}_j^t. \tag{6.14}$$

When the number of available sensors in $\mathcal{N}_i$ are different, the aggregated information scale is also different. There are two common ways to solve this issue: normalization and rescaling,

both of which can preserve the same scale of the expected value. Here we choose rescaling, because it can preserve the heterogeneity of information sent from one sensor to its different neighbors. We use a toy example in Figure 6.6 to help explain the difference. As we can see, when there is only one sensor available, all missing sensors will receive the same message under the normalization trick; but rescaling can avoid this problem because it preserves the column structure in messages sent from the same sensor. Thus, the message received by sensor i at step $t+1$ correspondingly becomes,

$$\mathbf{m}_i^{t+1} = \frac{|\mathcal{N}_i|}{|\{j \mid j \in \mathcal{N}_i, \| \mathbf{h}_j^t \| > 0\}|} \sum_{j \in \mathcal{N}_i, \|\mathbf{h}_j^t\|>0} \left(\tilde{\mathbf{D}}'_{ii} \tilde{\mathbf{D}}'_{jj}\right)^{-\frac{1}{2}} \tilde{\mathbf{A}}'_{ij} \mathbf{h}_j^t. \tag{6.15}$$

Gated Update Function. Here we use a GRU-based gating mechanism to automatically control information flow at each sensor. During message passing steps, the goals of missing sensors and available sensors are different. The missing sensors need to collect information related to recovering their own features, but also relay useful information to neighbors. For the available sensors, they actually only need to work as "relay nodes" to forward information to their missing neighboring sensors, because we will substitute output of available sensors by their original feature vectors later. In neural message passing, we can regard the passing steps as a special time dimension, where the newly received message is used as the external input at current step. Each sensor has an independent GRU implemented. Now the update function looks like,

$$\begin{aligned}
\text{Soft reset gate: } & G_{sr}^{t+1} = \sigma\left(\mathbf{W}_r \left[\mathbf{h}^t, m^{t+1}\right] + b_r\right), \\
\text{Hard reset gate: } & G_{hr}^{t+1} = 1 - \operatorname{sgn}\left(\| \mathbf{h}^t \|^2\right), \\
\text{Update gate: } & G_u^{t+1} = G_{hr}^{t+1} + \left(1 - G_{hr}^{t+1}\right) \sigma\left(\mathbf{W}_u \left[\mathbf{h}^t, m^{t+1}\right] + b_u\right), \\
\text{Current memory: } & \tilde{\mathbf{h}}^{t+1} = \tanh\left(\mathbf{W}_m \left[\mathbf{m}^{t+1}, G_{sr}^{t+1} \odot \mathbf{h}^t\right] + \mathbf{b}\right), \\
\text{Final memory: } & \mathbf{h}^{t+1} = \sigma\left(\left(\mathbf{\Theta}^{t+1}\right)^\top \left[G_u^{t+1} \odot \tilde{\mathbf{h}}^{t+1} + (1 - G_u^{t+1}) \odot \mathbf{h}^t\right]\right),
\end{aligned} \tag{6.16}$$

where $\operatorname{sgn}(x) = 1$ if $x > 0$, $\operatorname{sgn}(x) = 0$ if $x = 0$, and $\operatorname{sng}(x) = -1$ if $x < 0$. $\| \cdot \|^2$ is the L_2 norm. For notational simplicity, we remove the subscript for sensors here, but please remember each sensor has a separate gating module. The gating mechanism automatically learns how much information to preserve from the past, and what information to extract from the new message according to the learning objective. The only change we make to GRU is that we rename the original reset gate as a soft reset gate, and include a new hard reset gate. The hard reset gate evaluates whether the sensor representation at the previous step is null. If so, it will let the new sensor representation fully depend on the received message

(*i.e.*, set update gate as 1); otherwise, the update gate is used as its original definition.

Output Substitution. As we mentioned earlier, during message passing steps, the available sensors have to store information to be forwarded to their neighbors. The stored information is not related to reconstructing their own feature vectors. Such information pollution is inevitable at intermediate steps, but we can manually substitute the output of available sensors by their original feature vectors at the output layer. After all T rounds of message passing, the output at each sensor is,

$$\mathbf{h}_i^T = \text{sgn}\left(\left\|\tilde{\mathbf{h}}_i\right\|^2\right) \cdot \tilde{\mathbf{h}}_i + \left(1 - \text{sgn}\left(\left\|\tilde{\mathbf{h}}_i\right\|^2\right)\right) \cdot \mathbf{h}_i^T, \tag{6.17}$$

where $\|\cdot\|^2$ denotes the L_2 norm of the feature vector. We should be careful when using the output substitution because the reconstructed features of missing sensors may not be fully compatible with the original features of available sensors. This is why we train the neural network in an end-to-end manner so the remaining network components can adjust themselves accordingly. So far, we have introduced all components and design choices in the graph recovery module. Next, we describe how we design the loss function for the neural network equipped with the graph recovery module, and the corresponding training method.

6.4.3 Loss Function and Training Method

In this part, we describe the design of loss functions and training method for recognition models, equipped with the graph recovery module. The general philosophy is that we try to simulate different missing sensor situations during training to let the model automatically learn to adapt to and handle these situations.

Hybrid Training: In order to increase model robustness and improve its capability for handling missing sensors, we need to randomly drop features of some sensors during training. Vaizman *et al.* [199] preset a sensor missing probability p, where each sensor is independently dropped with probability p during training. However, according to our empirical observations, it is hard to find a value p that can effectively maintain model performance under different ratios of missing sensors during inference. If p is close to 0, it can effectively maintain model performance when most sensors are present, but it incurs severe performance degradation when most sensors are missing. In contrast, a large p value maintains model accuracy with few sensors, but can not maximally utilize all information when most sensors are present. Therefore, we propose a *hybrid training* method to tackle this problem. Instead of setting a single value of p, we heuristically choose a set of four probabilities $\mathcal{P} = \{0, 0.3, 0.6, 0.9\}$ in our training. The distorted samples under different

probabilities are concatenated into a new batch. Although we sacrifice some computational resources during such training, this one-time effort does effectively improve the balance of model performance across different test cases. Note that, model efficiency during inference is not affected.

Loss Function: The loss function of neural networks equipped with the graph recovery module consists of three parts: feature reconstruction loss, classification loss, and L_2 regularization.

Feature Reconstruction Loss L_R: Given the complete sensor feature matrix $\mathbf{H}$, the reconstruction loss is defined as the mean squared error (MSE) between the reconstructed sensor features and the original features for all missing sensors $\mathcal{M}$.

$$L_R = \frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} \left\| \mathbf{h}_i - \mathbf{h}_i^T \right\|^2. \tag{6.18}$$

Since we performed output substitution for available sensors at the output layer of the graph recovery module, there is no need to count the reconstruction loss for available sensors.

Classificatoin Loss L_C: For the classification loss, we choose the commonly used cross-entropy loss,

$$L_C = -\sum_{y \in \mathcal{Y}} p(y) \log q(y), \tag{6.19}$$

where $\mathcal{Y}$ is the labels, $p(y)$ and $q(y)$ denote the predicted class distribution and groundtruth label distribution.

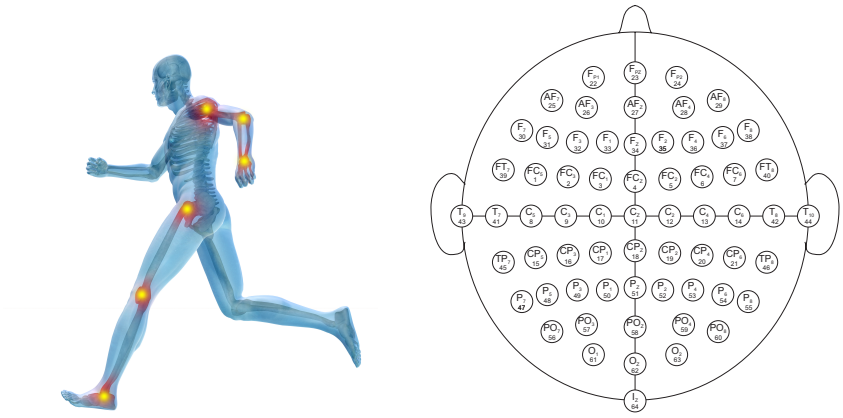
L_2 Regularization Loss L_N: In addition to the previous two learning losses, we add an L_2 regularization to prevent overfitting during training,

$$L_N = \sum_{\mathbf{w} \in \mathcal{W}} \|\mathbf{w}\|^2, \tag{6.20}$$

where $\mathcal{W}$ represents the set of all trainable parameters. Moreover, to cope with the hybrid training method introduced above, we add up the training loss under different missing sensor probabilities. Thus, the final loss function is,

$$L = \sum_{p \in \mathcal{P}} \alpha_p \left(\beta_R L_R^p + \beta_C L_C^p \right) + \beta_N L_N, \tag{6.21}$$

where α is a hyperparameter list related to each missing sensor probability, β is a hyperparameter list related to each loss component, and the superscript L^p specifies the loss corresponding to different missing sensor probabilities. Finally, we have to ensure that the whole



(a) Human Skeleton Network (b) EEG Electrode Network

Figure 6.7: Spatial topology networks we use in human activity recognition (HAR) and EEG-based motor-imagery recognition.

model is trained together in an end-to-end manner, because the *output substitution* step we use in graph recovery may lead to possible feature incompatibility problems if training the reconstruction module and classification model separately. We will empirically compare model performance between end-to-end training and separate training in Section 6.6 to validate this choice.

6.5 APPLICATIONS

In this section, we introduce two topology-aware multi-sensor IoT applications, where we can apply the graph recovery module to handle missing sensors. The corresponding backbone neural network and spatial topology we use in each application are also included.

6.5.1 Human Activity Recognition (HAR)

Human activity recognition (HAR) has been one of the most popular sensing tasks in the past decade. It is used in elderly/youth care [221], patient monitoring [160], and industry manufacturing assistance [222], among other applications. Here, we only consider activity recognition based on on-body inertial sensors. For professional monitoring and recognition systems, there are typically multiple Inertial Management Unit (IMU) sensors deployed at different body positions to capture full body movement; even in daily applications, we can combine the data from smart glasses, smart watches, smart phones, and smart chips in the shoes to infer user activity. Missing sensors do occur frequently in this scenario. For example, users may forget to wear the smart watch, or the phone is out of power. We want

to maximize the recognition accuracy no matter what devices are available, but at the same time we prefer that only one model be loaded into the system. In our implementation, we construct the sensor topology according to their corresponding joint connection relationships according to the human skeleton structure, as shown in Figure 6.7(a). The intuition behind it is that human skeleton limits the degrees of freedom of whole body motion, because directly connected joints satisfy certain physical constraints on their movement patterns. In the graph recovery module, such correlations are reflected in the learned adjacency weight matrix. For the backbone network, we choose a variant of the DeepSense framework proposed by Yao *et al.* [148]. For the input, we first divide the sensing signal of each sensor into T time intervals, and then get the frequency spectrum of each interval. The design choices corresponding to the general learning framework in Figure 6.4 are as follows: In sensor feature extraction, we use a three-layer convolutional module to extract individual features of each sensing modality, followed by another three-layer convolutional module to fuse information from different sensors at the same body position. Sensor fusion across all body positions is a mean pooling layer, instead of complex attention mechanisms [40, 159]. Finally, we use a network consisting of two Gated Recurrent Unit (GRU) layers and one fully connected (*i.e.*, dense) layer to work as the classification model.

6.5.2 Electroencephalogram (EEG)-based Motor-imagery Recognition

The Electroencephalogram (EEG) test [215] is a test used to evaluate the electrical activity in the brain. During the test, a set of electrodes are placed along the scalp to measure voltage fluctuations resulting from brain activity. Specifically, we utilize the EEG records for user motor-imagery recognition. EEG data is inherently noisy because EEG electrodes also pick up unwanted electrical physiological signals, such as the electromyogram (EMG) from eye blinks and muscles on the neck [223]. There is usually a manual channel/electrode selection process in data preprocessing, where the selected electrodes can be different case by case. This can also be regarded as a special "missing sensor" scenario. We need the graph recovery module for feature reconstruction to guarantee different selected electrodes can still use the same recognition model. Meanwhile, there is an internationally recognized method to describe and apply the local of scalp electrodes in the context of an EEG exam [212]. In Figure 6.7(b), we show the topology structure of electrodes as per the international 10-10 system that is also used in our evaluation. We manually connect the neighboring nodes in this network to form the electrode topology to feed into the graph recovery module. The input data format is similar to what we use in the HAR task, except that we only have one sensing modality here. Our backbone network is based on the design of Qiao *et al.*

in [198]. The sensor fusion in both spatial and temporal dimensions is performed together by a convolutional module. At the bottom feature extraction, we use a three-layer convolutional module to extract features for each electrode within every time interval. On top of the extracted local features, the sensor fusion module regards the aggregated features as a special spatial-temporal "image", where the electrode positions and time intervals are chosen as two image dimensions. We use another three-layer convolutional module to aggregate information across different sensor positions and time intervals. Finally, a fully connected layer is stacked on top of the sensor fusion module before we output the final prediction results.

6.6 EVALUATION

In this section, we empirically evaluate the reconstruction performance of the proposed graph recovery module on two publicly available human activity recognition (HAR) datasets, and an EEG-based motor-imagery recognition dataset. We first introduce the experimental setup, the datasets we use along with their corresponding preprocessing procedures, and the baseline algorithms we are comparing with. We then show a direct comparison of model classification performance with baseline models. We also compare them with expert models trained specially under two end-cases; namely, sensor drop probability $p = 0$ and $p = 0.9$. Next, we compare the effect of different training regimes on GraphRecovery. After that, we perform a quantified evaluation of the reconstructed features by different algorithms against the original features, to demonstrate the contribution of the graph recovery module to the reconstruction. Finally, we report the time and energy efficiencies on a commodity IoT device; a Raspberry Pi 3 Model B.

6.6.1 Experimental Setup

All models evaluated in this work are trained with Tensorflow 1.14 [29] on a workstation equipped with an Intel i9-9960X processor, 64GB memory, and four NVIDIA RTX 2080 Ti GPUs. The model is optimized by the ADAM algorithm [166] with a learning rate that varies from $1e^{-3}$ to $1e^{-5}$ across different datasets, while $\beta_1 = 0.5$ and $\beta_2 = 0.9$. We add a batch normalization layer and a dropout layer after each convolutional layer to stabilize the training process and prevent overfitting. Training batch size is set as 64. For the hybrid training, we take the unweighted sum of missing sensor probabilities (*i.e.*, $\alpha_p = 0.25, \forall p \in \mathcal{P}$). For other hyperparameters in the loss function, $\beta_C = 1$, $\beta_N = 5e^{-4}$, while β_R is tuned differently on each dataset.

Table 6.1: Statistical Summary of Selected Datasets.

Dataset	Classes	Subjects	Sensors	Positions	Sample Length	Sample Frequency	#Samples	#folds
REALDISP	33	17	3	9	3 sec	50 Hz	5225	4
RealWorld-HAR	8	15	3	7	3 sec	50 Hz	21257	5
EEG-MMID	3	109	1	64	4.1 sec	160 Hz	37951	4

During the evaluation, we assume that each sensor is offline independently with a probability p, where p varies from 0 to 0.9. We try to emulate different missing sensor cases to test model robustness in each case. Sensing readings of missing sensors are set as 0 all the time. Moreover, all models are evaluated under a leave-one-user-out scenario with k-fold cross validation. Specifically, we divide the involved subjects in each dataset into k groups, and choose all samples of one user group as testing data every time, whereas others are used for training. There is no overlap among the user groups. When evaluating algorithm performance for the same user population, we use 90% of the subjects from all groups as training data, and use the remaining 10% subjects as the validation data. All user groups are tested in a cross-validation manner, then their average results are presented and analyzed in the following subsections. The choice of k in each dataset is shown in Table 6.1.

6.6.2 Datasets and Data Preprocessing

We first briefly introduce our data preprocessing steps before introducing the selected datasets and their corresponding statistics. We summarize the related data statistics of each dataset in Table 6.1.

Suppose the sensing signals $\mathbf{X}$ are collected from $|\mathcal{V}| = N$ positions for a fixed time period (*e.g.*, 3 seconds), where S sensor types are deployed at each position. Each sensor can have d dimensions. For example, an IMU sensor typically has three sensing modalities (*i.e.*, accelerometer, gyroscope, and magnetometer), each of which has three dimensions, x, y, and z. Instead, the electrode only has one sensing modality with one dimension. Different sensing modalities are upsampled and downsampled into a unified sampling rate. The signals of each dimension are divided into T fixed-length and non-overlapped time intervals. We further perform a Fourier transform to each interval to extract their frequency domain spectra, which have been proved to be more informative than pure time domain representations [163]. After preprocessing, the input fed into the model should have a shape of $\mathbf{X} \in \mathbb{R}^{T \times N \times S \times d \times 2f}$, where T represents time intervals, N denotes sensor positions, S denotes sensing modalities, d is the sensor dimensionality, and $2f$ is the number of spectral samples with f frequency magnitude and phase pairs within each interval. Such time-frequency representation preserves both time domain order information and frequency domain pattern information simultaneously.

REAListic sensor DISPlacement Activity Recognition Dataset (REALDISP) [207]: This dataset has been originally collected to investigate the effects of sensor displacement in the activity recognition process in real-world settings. They cover 9 different body positions and 33 activity classes. An IMU sensor is deployed on each body position. Different sensor placement situations are considered, including ideal-placement, self-placement (*i.e.*, decided by user), and induced placement (*i.e.*, with intentional sensor rotations and translations). In our evaluation, we use data collected under all sensor placement situations. A total of 17 users involved in data collections are divided into 4 folds in cross-validation. The data are sampled at 50 Hz. We divide the 3 seconds samples into 10 time intervals, where each interval contains 15 readings.

RealWorld Human Activity Recognition Dataset (RealWorld-HAR) [156]: This dataset covers 8 activities from 15 subjects on 7 body positions, with an IMU sensor deployed on each position. The subjects are divided into 5 folds for cross-validation. The data are sampled at 50 Hz. Similarly, we divide the 3 second samples into 10 time intervals, where each interval contains 15 readings.

EEG Motor Movement/Imagery Dataset (EEG-MMID) [208]: In this dataset, 109 subjects performed different motor/imagery tasks while a 64-channel EEG was recorded using the BCI2000 system. 109 subjects are divided into 4 folds for cross-validation. Three annotated classes are rest, onset of motion (real or imagined) of left/both fist(s), and onset of motion (real or imagined) of right fist or both feet. The electrodes are placed according to the 10-10 system shown in Figure 6.7(b). The voltage is sampled at 160 Hz. We divide the 4.1 second samples (one action time in this dataset) into 16 time intervals, where each interval contains 41 readings.

6.6.3 Baseline Algorithms

In this part, we briefly introduce the baseline models we compare in our experiments. They include three types of networks: backbone networks, baseline algorithms for handling missing sensors, and variants of graph recovery.

- **Backbone Networks (BackboneNet, BN):** In order to show how different feature reconstruction algorithms can improve model robustness when encountering missing sensor situations, we compare their performance with the backbone network under each missing sensor probability. As we mentioned in Section 6.5, we use DeepSense [148] as the backbone network for all HAR datasets, and use a CNN network [198] for the EEG dataset.

- **Dropout Training Network (DropoutNet-Single, DN-Sin) [199]:** Instead of setting up an extra module to reconstruct features for missing sensors, we keep using the backbone network in this algorithm. The only difference is that we randomly drop some sensors during the training to make the neural network learn to adapt to different situations. We follow Vaizman *et al.* in their paper [199] to set the missing sensor probability $p = 0.5$, which attains a relatively good balance between the two ends (*i.e.*, $p = 0$ and $p = 0.9$). This model follows the design in their original paper.

- **Dropout Training Network with Hybrid Training (DropoutNet-Hybrid, DN-Hyb):** This is a variant of DropoutNet with hybrid training. Instead of choosing a single sensor drop probability, we use the same hybrid sensor drop probabilities as GraphRecovery during the training. By comparing the DropoutNet-Single with DropoutNet-Hybrid, we can see how hybrid training can affect the model with no feature reconstruction module. By comparing DropoutNet-Hybrid with GraphRecovery, we can have a more straightforward understanding on the contribution of the proposed graph recovery module in improving model robustness against missing sensors.

- **Graph Autoencoder (GAE) [201, 203]:** In this algorithm, we use an autoencoder to reconstruct missing sensor features. Both the encoder and decoder are implemented by stacked graph convolutional layers, and a bottleneck fully connected layer is put between the encoder and decoder. During the training, we first train a backbone network based on all sensor readings, and then train the reconstruction module on top of the fixed classification model. Mean squared error (MSE) loss between the reconstructed features and the original features is used as the optimization objective when training the reconstruction module. The main advantage of this algorithm is that it doesn't need to retrain the whole neural network. We only need to train a separate feature reconstruction module based on the Encoder-Decoder architecture, on top of the existing backbone network.

- **Graph Recovery with Separated Training (GR-Sep):** In this variant, the network architecture is the same as what we use in GraphRecovery. The only difference is that we train the classification network and feature reconstruction network separately here. This variant is used to verify the possible compatibility problems between the feature reconstruction module and the classification module, so that we can understand the importance of end-to-end training in neural network equipped with graph recovery.

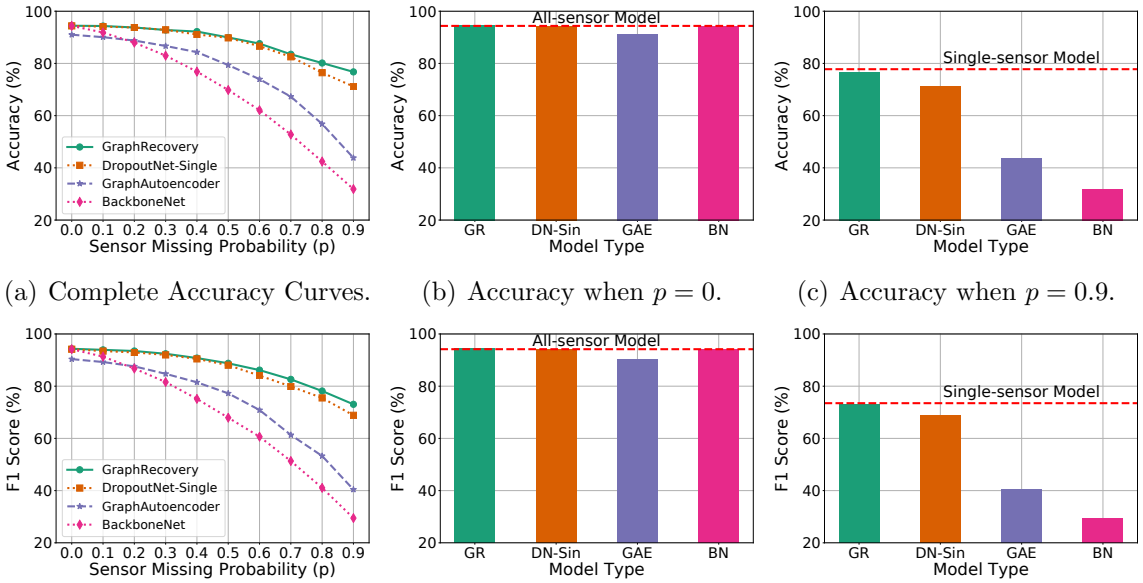
- **Graph Recovery with Single Training (GR-Single):** The only difference between

this variant and GraphRecovery is that we do not use the proposed hybrid training method here. Instead, we set the missing sensor probability $p = 0.5$ uniformly. By comparing GR-Single and GR, we can see how hybrid training can help the network attain a better balance under different missing sensor probabilities, especially when $p > 0.5$.

6.6.4 Quantitative Evaluation of Classification Performance

In this subsection, we present and analyze the classification results of GraphRecovery and the aforementioned baseline algorithms. We choose accuracy and F1 score as the evaluation metrics here. Accuracy is an intuitive global performance measure for the model, while F1 score is better at dealing with imbalanced class distributions. To evaluate the contribution of the graph recovery module, we pay more attention to the model performance degradation under different missing sensor situations, compared to the situation where all sensors are available. In some figures, for notational simplicity, we use abbreviations to represent models. GR represents GraphRecovery, which is the network equipped with the graph recovery module. GAE represents GraphAutoencoder, which is the network equipped with a GCN-based autoencoder module. We use DN-Sin to represent DropoutNet-Single, where no hybrid training is applied. The DropoutNet-Hybrid with hybrid training will also be evaluated and discussed later. Generally, applying hybrid training to DropoutNet makes its performance worse, so we don't consider it here. Specific reasons will be analyzed in the next subsection. We use BN to denote the backbone network. In addition to showing the complete accuracy curve changing with p values, we also separately compare the model accuracy when $p = 0$ and $p = 0.9$, with the models specifically trained for these situations, which we also call the *expert models*. When $p = 0$, the expert model is trained with all sensors; while when $p = 0.9$, the expert model is trained with data samples extracted from all possible combinations of 10% sensors. The performance of expert models are shown in red dashed lines with corresponding annotations in these figures. They are used as performance "upper bound" for compared algorithms. Next, results of each dataset are analyzed one by one.

REALDISP: The classification results on the REALDISP dataset are shown in Figure 6.8. Let's start from $p = 0$ in Figure 6.8(b). One basic requirement of the missing sensor reconstruction module is that it should not degrade performance under when all sensors are present; we have to maintain the original backbone model performance when all sensors are available. Both GraphRecovery and DropoutNet-Single can attain similar performance as the BackboneNet, with respect to both accuracy and F1 score. However, GraphAutoencoder has a clear performance drop, which means that the reconstructed features by the autoen-

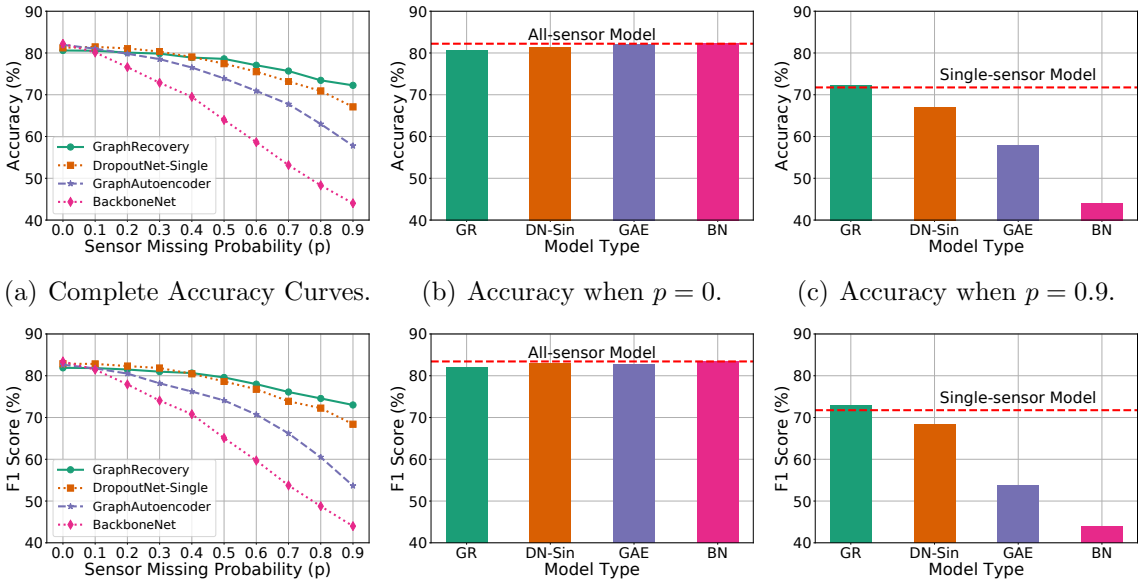


(a) Complete Accuracy Curves. (b) Accuracy when $p = 0$. (c) Accuracy when $p = 0.9$.

(d) Complete F1 Score Curves. (e) F1 score when $p = 0$. (f) F1 score when $p = 0.9$.

Figure 6.8: Performance comparison under different sensor missing probabilities on REALDISP.

coder can not perfectly fit into the original classification network, even when all sensors are present. According to the curves in Figure 6.8(a) and Figure 6.8(d), all three feature reconstruction algorithms (*i.e.*, graph recovery, dropout, and graph autoencoder) are effective in improving model robustness compared to the BackboneNet. GraphRecovery is the best model among the three models. We also find that the performance gap between GraphRecovery and DropoutNet-Single increases along with the increasing p value, which means that the model with graph recovery module is more robust against missing sensors. Meanwhile, they are significantly better than the GraphAutoencoder and BackboneNet. When $p = 0.9$, the performance of GraphRecovery is close to the expert model, while there is a 6.62% accuracy loss on DropoutNet. We can understand dropout from the perspective of an ensemble mechanism [224]. If we set $p = 0.5$ during the training, the number of available sensors in most "dropout" training samples is around 50%. During inference, the model takes the voting / average result of all partial sensor combinations. When the value of p is small, the voting results are summarized from diverse partial sensor combinations. Thus, the ensemble purpose by dropout is effective, leading to near-optimal performance in DropoutNet. On the contrary, when the p value is close to 1, most sensors are missing, and their features are simply set as 0. This leads to two effects: First, each partial sensor combination contains much less than 50% available sensors, so they can't attain similar performance as the training samples. Second, the diversity of sensor combinations becomes much lower, so



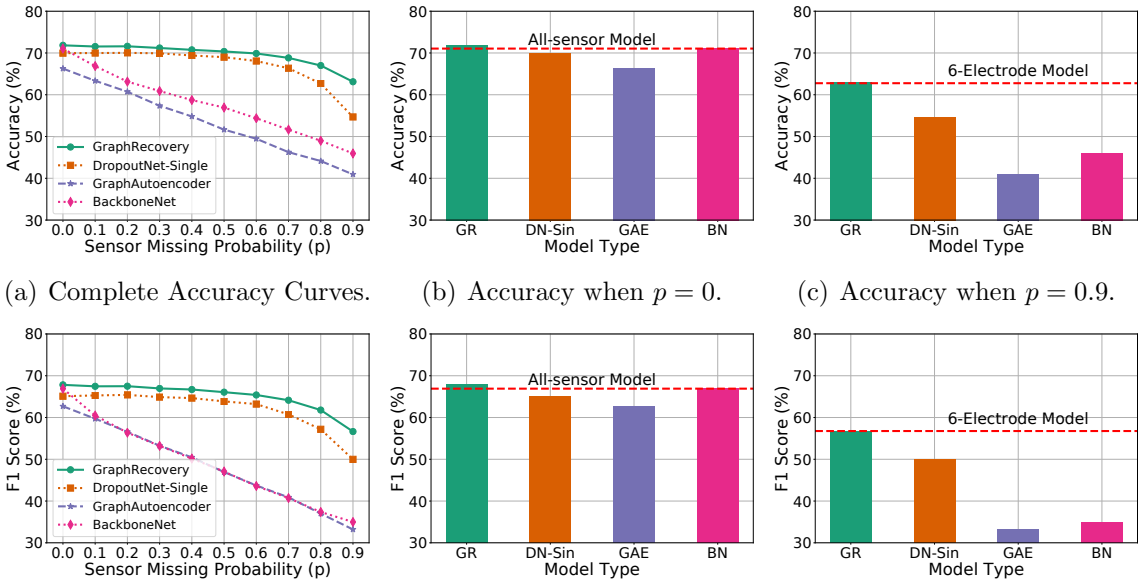
(a) Complete Accuracy Curves. (b) Accuracy when $p = 0$. (c) Accuracy when $p = 0.9$.

(d) Complete F1 score Curves. (e) F1 score when $p = 0$. (f) F1 score when $p = 0.9$.

Figure 6.9: Performance comparison under different sensor missing probabilities on RealWorld-HAR.

that the ensemble purpose can no longer be achieved. Therefore, a larger performance drop can be observed in DropoutNet-Single when p approaches 1. Instead, GraphRecovery and GraphAutoencoder perform classification based on features from all sensors, where the features for missing sensors are reconstructed by their feature reconstruction modules. We also observe that the accuracy loss between GAE and the expert model is large, which indicates that the reconstructed features can not effectively utilize the classification network trained on original features.

RealWorld-HAR: The corresponding results are shown in Figure 6.9. Some observations are similar to the results in REALDISP, so we do not repeat them. There are also a few new points we need to mention here: First, the accuracy gap between graph recovery and dropout training when $p = 0.9$ is even larger, which demonstrates their different capabilities in increasing model robustness. The feature reconstruction module in GraphRecovery preserves the logic of sensing inference in a unified way. Second, as shown in Figure 6.9(b), there is a small performance loss ($\sim 1.5\%$) on GraphRecovery compared to the BackboneNet when $p = 0$. After investigation, the reason we find is that the dataset itself is noisy and sensitive. We can not tune the network to be optimal at both ends (*i.e.*, both $p = 0$ and $p = 0.9$). Thus, hybrid training causes a negative effect on the classification module, when p is large. As we will show in the next subsection, this performance drop at $p = 0$ doesn't exist in GraphRecovery-Single, where no hybrid-training is applied. Third, the performance of



(a) Complete Accuracy Curves. (b) Accuracy when $p = 0$. (c) Accuracy when $p = 0.9$.

(d) Complete F1 score Curves. (e) F1 score when $p = 0$. (f) F1 score when $p = 0.9$.

Figure 6.10: Performance comparison under different sensor missing probabilities on EEG-MMID.

GraphRecovery is even slightly better than the single-sensor expert model in Figure 6.9(c) and Figure 6.9(f). This phenomenon is reasonable because at least one sensor is available in each data sample, so that the expected available sensors in the testing data is slightly more than 1. Instead, in the single-sensor expert model, we only use the single-sensor data samples for training and testing.

EEG-MMID: In the EEG-MMID dataset, the number of sensor positions is significantly larger than the sensor positions in both HAR datasets (*i.e.*, 64 vs. 7 or 9). The results are summarized in Figure 6.10. We see a slight advantage of BackboneNet over DropoutNet-Single when $p = 0$ in Figure 6.10(b) and 6.10(e), which indicates that a large sensor dropout probability (*i.e.*, $p = 0.5$) during training can also have a negative effect on model performance, because the global feature patterns from most sensor positions are ignored during training of DropoutNet-Single. In the EEG task, the global information aggregated from all sensor positions is more distinguishable towards output classes, than the ensemble decision of partial sensor groups. After combining the results from all 3 datasets, we can safely conclude that both GraphRecovery and DropoutNet can effectively maintain model performance when most sensors are available, with only possible slight deviation from the BackboneNet. GAE does not always attain comparable performance. For example, in EEG-MMID (Figure 6.10(b) and 6.10(e)), there is a 5.29% accuracy drop and a 5.13% F1 score drop on GAE compared to the expert model when $p = 0$. This validates the importance of end-to-end

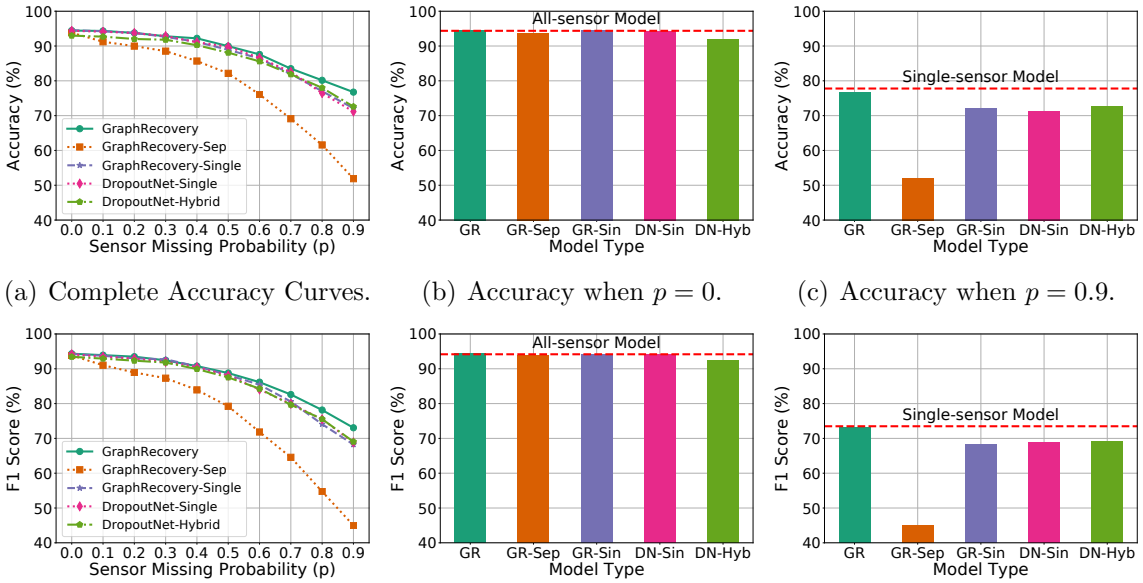
training of all network components when dealing with missing sensors. Moreover, the advantage of GraphRecovery over DropoutNet is more significant in EEG-MMID. According to Figure 6.10(a) and 6.10(d), when p increases, GraphRecovery maintains a clearly better performance than DropoutNet. Therefore, we empirically show that the graph recovery module is better than the baseline models in improving model robustness against missing sensors.

After analyzing each dataset results individually, we summarize the conclusions that are found commonly in all datasets.

- Graph recovery is the best among the three compared algorithms at improving general model robustness against missing sensor situations, which attains a performance close to the expert models at both ends (*i.e.*, $p = 0$ and $p = 0.9$).
- The advantage of the proposed graph recovery over baseline algorithms is more significant when applied to IoT applications with larger scale spatial sensor networks (*i.e.*, EEG vs. HAR tasks).
- In order to handle missing sensors, it is important to train the feature reconstruction module and backbone classification network together in an end-to-end manner. Otherwise, the reconstructed features may not be able to properly utilize the pre-trained classification network, even under the situation where all sensors are available (*i.e.*, GraphAutoencoder in RealWorld-HAR and EEG-MMID).

6.6.5 Ablation Study on Training Regimes

In this part, we perform an ablation study on the proposed graph recovery module with respect to different training regimes. We compare GraphRecovery with its two variants, GraphRecovery-Sep (GR-Sep), and GraphRecovery-Single (GR-Single). In GR-Sep, instead of training the whole network in an end-to-end manner, we first train a backbone network for performing classification based on all sensors, and train a separate graph recovery module to reconstruct the features of missing sensors. By comparing GR with GR-Sep, we can have a direct understanding on how the compatibility between reconstruction module and classification network can affect classification performance on reconstructed features. In GR-Single, we give up the hybrid training, but only use a fixed missing sensor probability $p = 0.5$ during the training. By comparing GR with GR-Single, we can see how the hybrid training method can help the model find a better performance balance under different missing sensor probabilities. In addition, we also include DropoutNet-Single and DropoutNet-Hybrid



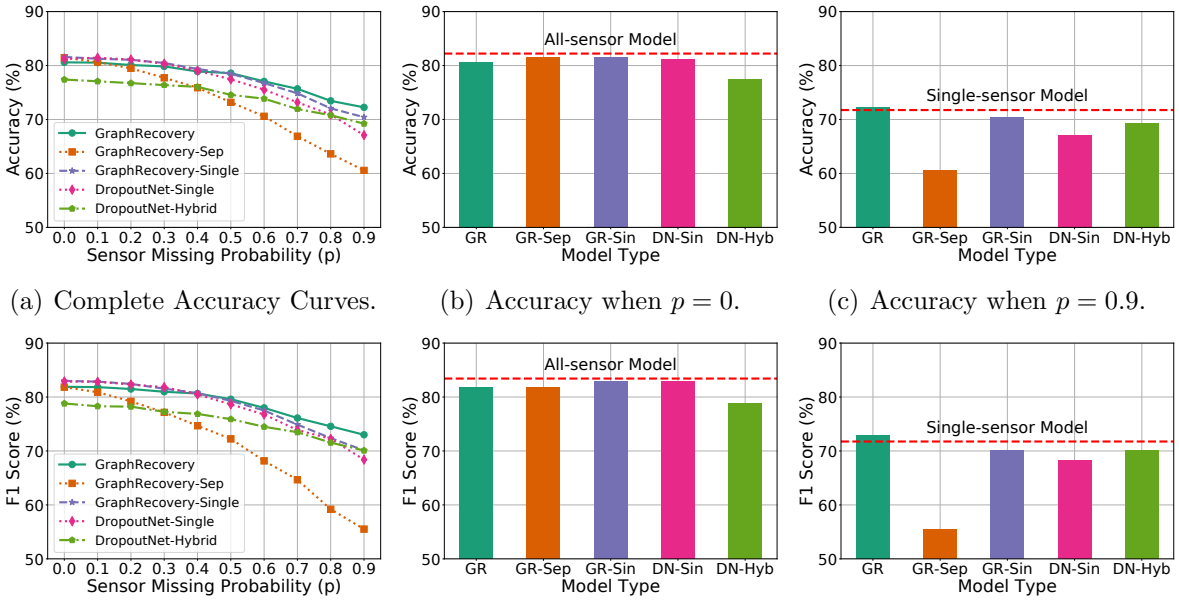
(a) Complete Accuracy Curves. (b) Accuracy when $p = 0$. (c) Accuracy when $p = 0.9$.

(d) Complete F1 Score Curves. (e) F1 score when $p = 0$. (f) F1 score when $p = 0.9$.

Figure 6.11: Ablation study on REALDISP.

here. By comparing these two models, we can see that hybrid training can also have a negative impact on model performance, if not utilized appropriately. The improvement of GraphRecovery over GraphRecovery-Single not only comes from hybrid training, but also from the feature reconstruction capability of the graph recovery module. All results are summarized in Figure 6.11, 6.12, and 6.13, where we choose the same way as before to present the results. We show both the complete accuracy and F1 score change curves, and specific performance comparisons when $p = 0$ and $p = 0.9$. Besides, the expert models for each condition (*i.e.*, $p = 0$ and $p = 0.9$) are still shown in red dashed lines with special annotations.

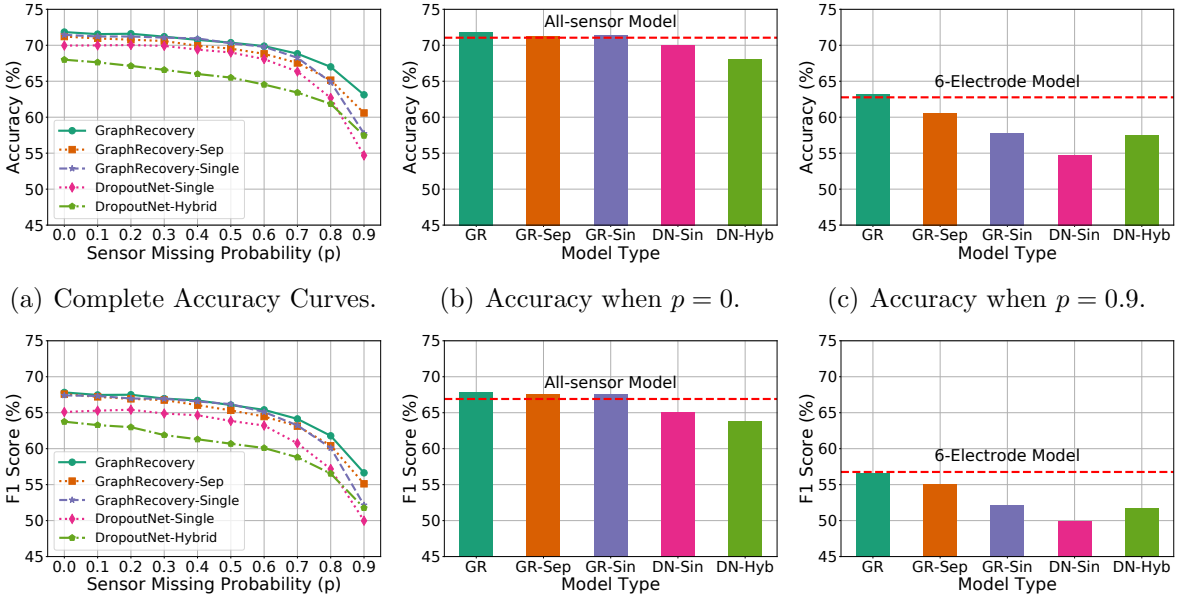
To partially resolve the compatibility issue between the feature reconstruction module and classification module in GR-Sep, we add a *knowledge distillation loss* [225] in addition to the Mean Squared Error (MSE) loss when training the graph recovery module. The intuition is to utilize the pre-trained classification network to give the feature reconstruction module additional supervision. Intuitively speaking, the learning objective of the feature reconstruction module is not only to perfectly reconstruct the original features, but also to achieve similar inference results as original features. During knowledge distillation, the parameters in the classification network are not updated. We choose the fully connected layer before the output layer as the target for knowledge distillation. Assume the sub-network between the feature reconstruction module and the target layer is C. The reconstruction



(a) Complete Accuracy Curves. (b) Accuracy when $p = 0$. (c) Accuracy when $p = 0.9$.

(d) Complete F1 Score Curves. (e) F1 score when $p = 0$. (f) F1 score when $p = 0.9$.

Figure 6.12: Ablation study on Realworld-HAR.



(a) Complete Accuracy Curves. (b) Accuracy when $p = 0$. (c) Accuracy when $p = 0.9$.

(d) Complete F1 Score Curves. (e) F1 score when $p = 0$. (f) F1 score when $p = 0.9$.

Figure 6.13: Ablation study on EEG-MMID.

loss becomes,

$$L_R = \frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} \left\| \mathbf{h}_i - \mathbf{h}_i^T \right\|^2 + \gamma \left\| C(\mathbf{H}) - C(\mathbf{H}^T) \right\|^2, \tag{6.22}$$

where γ is a hyperparamter to tune the weights between reconstruction loss and knowledge

distillation loss.

We first compare GraphRecovery with GraphRecovery-Sep. When $p = 0$, the performance of GraphRecovery and GraphRecovery-Sep are similar for all three datasets. From the accuracy curves, we can see that the performance gap between GR and GR-Sep increases with p increases, especially for HAR tasks (*i.e.*, REALDISP, and RealWorld-HAR). The accuracy gap can even go up to 24.85% when $p = 0.9$ in REALDISP (*i.e.*, shown in Figure 6.11(c)). This is due to the fact that the reconstructed features can not perfectly fit back into the classification network, trained on original sensor features. Therefore, our conclusion is that, the separated training can not replace end-to-end training in our problem setting, even if we add the knowledge distillation loss to guide the training of the reconstruction module.

We next compare GraphRecovery with GraphRecovery-Single. When $p = 0$, GR and GR-Single attain similar performance for both REALDISP and EEG-MMID dataset, while GR-Single is slightly better than GR for RealWorld-HAR. As we mentioned earlier, RealWorld-HAR dataset is generally noisy and sensitive. Therefore, it is hard to tune GR to obtain optimal results at both ends (*i.e.*, $p = 0$ and $p = 0.9$). The training difficulty of GR apparently is higher than GR-Single, since it has to deal with feature reconstruction at different probabilities. After comparing the accuracy and F1 curves of GR and GR-Single, we find that their performance is similar when p is small; more specifically, when $p \leq 0.5$. When p continues to increase, their performance gap starts to become larger. There is a clear performance drop in GR-Single when $p \geq 0.7$, which means the situations where most sensors are missing are not well handled by GR-Single. The single missing sensor ratio in GR-Single leads to an unbalanced feature reconstruction capability under different p values. This problem is more severe in the EEG-MMID task, where GR-Single has an even worse performance than GR-Sep at $p = 0.9$. Therefore, we have empirically shown the benefits of leveraging the hybrid training method (*i.e.*, apply multiple missing sensor ratios simultaneously) in GR. It emulates different sensor drop probabilities during training to help the model find a better balance in its reconstruction capabilities under different cases. The model performance at the right end (*i.e.*, $p = 0.9$) is lifted effectively, without significantly sacrificing the performance at the left end (*i.e.*, $p = 0$). Though it takes a relatively longer training time than GR-Single, this one-time effort does not affect the time efficiency during the inference stage.

However, when we compare DropoutNet-Hybrid and DropoutNet-Single, the benefits of hybrid training don't exist anymore. In all three datasets, the performance gain of DropoutNet-Hybrid to DropoutNet-Single at $p = 0.9$ is obtained at the cost of sacrificing the model performance in most cases (*i.e.*, $p \leq 0.7$). Following the previous understanding of dropout from the ensemble learning perspective, the involvement of too high dropout

probabilities (*i.e.*, 0.6 and 0.9 in hybrid training) is harmful because they are relying on the voting result of very local information pieces. For example, if we have 10 sensor positions in total, setting $p = 0.9$ during training means that the model inference result is the voting of 10 single-sensor models, where too much global multi-sensor information is lost. The result is that the model trades its performance at one end (*i.e.*, $p < 0.7$) for the possibly minor improvement at the other end ($p = 0.9$). It has been agreed in machine learning literature [224] that for DropoutNet, $p = 0.5$ is an empirically good choice, because it not only gets the highest variance in neuron (sensor) combinations, but also maintains most of the crucial information. Therefore, we can safely conclude that the success of hybrid training in GraphRecovery also comes from the strong feature reconstruction capabilities in the graph recovery module.

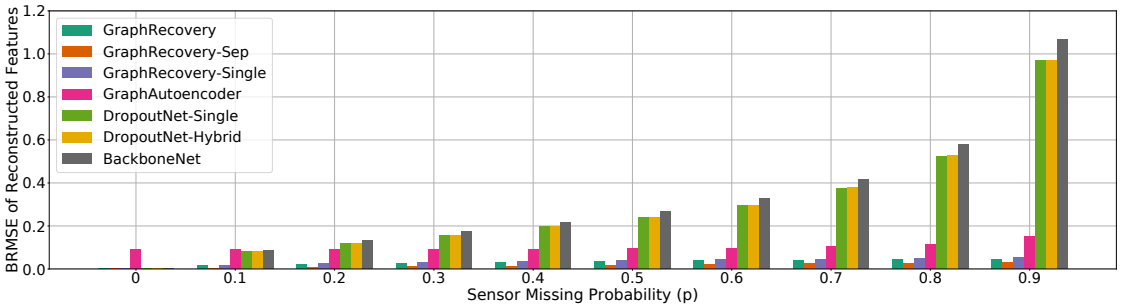
6.6.6 Feature Reconstruction Quantification

In this part, we compare the reconstructed features of different models against the original features under different missing sensor ratios. We make sure the backbone networks of different algorithms are completely the same, including their activation functions at each layer. Mean squared error (MSE) or root mean squared error (RMSE) is a commonly used metric to estimate the similarity between feature vectors in a latent space [226, 227]. Here, the feature reconstruction error is evaluated using the bit-based root mean squared error (BRMSE) between the reconstructed features and the original features, which is defined as,

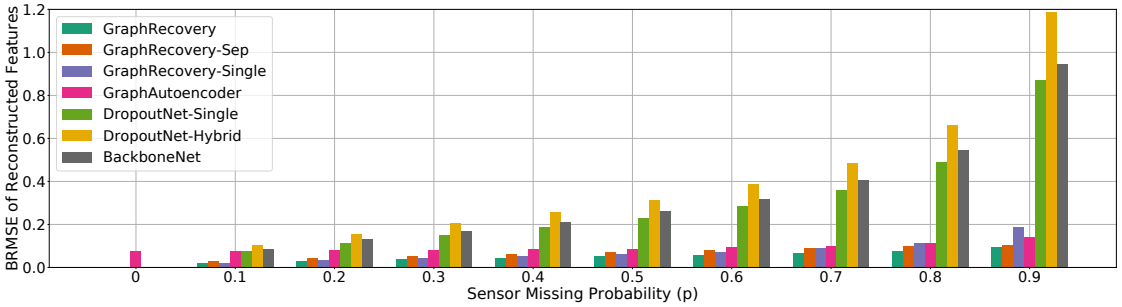
$$BRMSE = \sqrt{\frac{\sum_{x \in \mathcal{X}_T} \left\| \hat{\mathbf{h}}_x - \mathbf{h}_x \right\|^2}{|\mathcal{X}_T| \cdot |\mathbf{h}_x|}}, \tag{6.23}$$

where $\mathcal{X}_T$ represents the test dataset, h_x represents the original features of data sample x before the feature reconstruction layer, and $\hat{\mathbf{h}}_x$ is the reconstructed feature vector of the same data sample. Please note that we divide the testing dataset size and the dimension of feature vector at the denominator, so that BRMSE is not affected by the feature dimension as root mean squared error (RMSE).

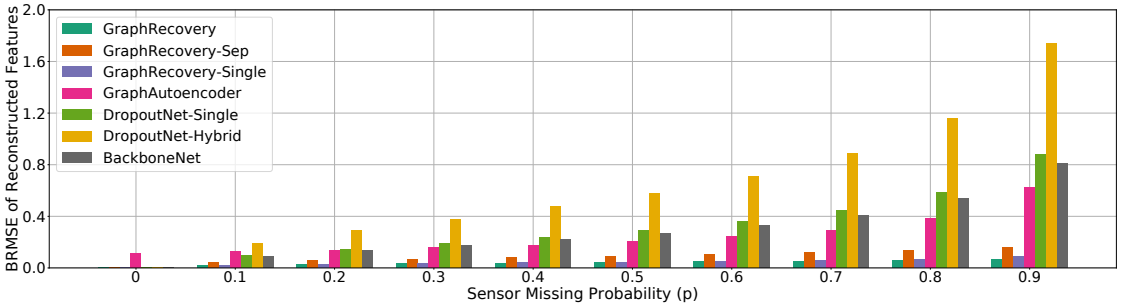
The feature reconstruction results are given in Figure 6.14. Since DropoutNet and BackboneNet don't have a dedicated feature reconstruction module, their reconstruction losses grow significantly with increasing p. GraphAutoencoder is much better than the above three baselines. Moreover, all three versions of the GraphRecovery model have a clearly lower reconstruction loss than GraphAutoencoder. To fairly compare the reconstruction performance of the graph recovery module and the autoencoder, we can look at the recon-



(a) REALDISP.



(b) Realworld-HAR.

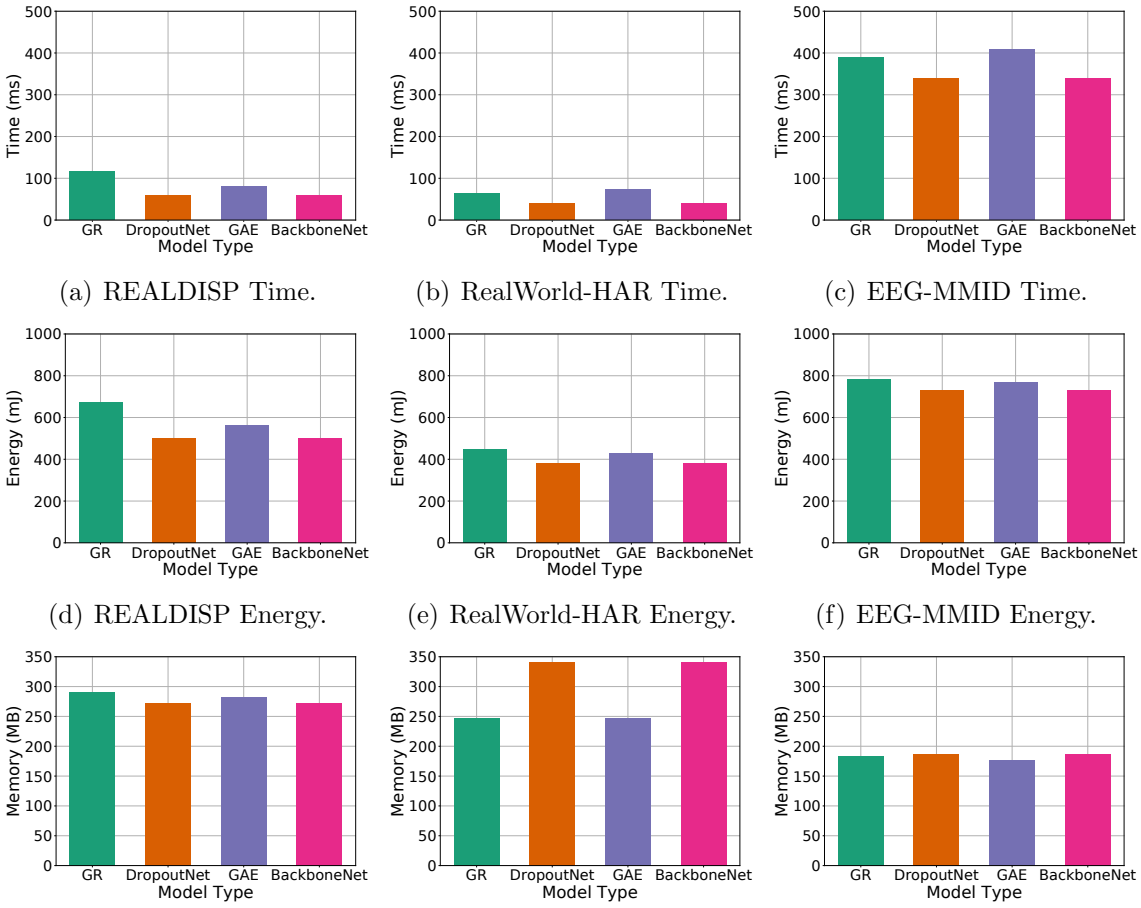


(c) EEG-MMID.

Figure 6.14: Feature reconstruction error quantification on REALDISP, RealWorld-HAR, and EEG-MMID.

struction loss between GraphRecovery-Sep and GraphAutoencoder, because their feature reconstruction modules are both trained separately on top of the same backbone network. In all three datasets, GraphRecovery-Sep beats GraphAutoencoder by presenting a lower reconstruction loss. When applied in large scale networks (*i.e.*, EEG-MMID), the reconstruction loss of GraphAutoencoder is several times higher than GraphRecovery-Separate. The conclusion here is that the graph recovery module is a better design in reconstructing the features of missing sensors.

When comparing the different variants of GraphRecovery, we found that GraphRecovery-Single generally shows similar reconstruction loss as GraphRecovery at lower sensor drop



(a) REALDISP Time. (b) RealWorld-HAR Time. (c) EEG-MMID Time.

(d) REALDISP Energy. (e) RealWorld-HAR Energy. (f) EEG-MMID Energy.

(g) REALDISP Memory. (h) RealWorld-HAR Memory. (i) EEG-MMID Memory.

Figure 6.15: Time, energy and memory consumption of compared algorithms on each dataset.

probabilities, while shows high reconstruction loss at higher sensor drop probabilities. This result is in accordance with the model classification performance. If we compare GraphRecovery and GraphRecovery-Sep, we can see that a lower reconstruction loss doesn't necessarily indicate a higher model performance. For example, in REALDISP, GraphRecovery-Sep always has a lower reconstruction loss than GraphRecovery, but its model accuracy and F1 score is significantly lower than GraphRecovery. This is because the classification network in GraphRecovery is also dynamically adjusted for the reconstructed features during training, leading to a better overall classification performance.

6.6.7 Time, Energy and Memory Efficiency

In this subsection, we report the time, energy and memory efficiencies of the graph recovery module on top of the backbone networks when they are deployed on commodity IoT devices. The experiments are conducted on Raspberry Pi 3B, which is powered by a quad core 1.2

Table 6.2: Backbone network sizes.

Dataset	REALDISP	RealWorld-HAR	EEG-MMID
Backbone network	DeepSense	DeepSense	CNN
#Layers	10	10	7
#FLOPs (in million)	432.842	263.309	3920.450

GHz Broadcom BCM2837 64bit CPU with 1 GB RAM. For all the models, we only use on-chip CPU for inference. Every model is preloaded to the IoT device before the experiment, and any unnecessary applications and services that may interfere with model computation are closed in advance. Time and memory are measured by the built-in modules, while the energy consumption is measured by an external Monsoon High Voltage Power Monitor [168]. We independently run each model on each dataset for 20 times, and take the average time and energy result. Before the testing, one warm-up run is performed.

The corresponding results are summarized in Figure 6.15. As we can see, since DropoutNet doesn't include any additional components compared to BackboneNet, they always exhibit the same time, energy and memory consumption on every dataset. On the contrary, extra time and energy cost are caused by GAE and GR models. Between these two algorithms, GAE is more efficient than GR in most cases. Although the encoder-decoder structure in GAE typically has more layers than GR, the included gating mechanism in GR becomes an efficiency bottleneck. The reason why we use GRU to design the gating mechanism, in lieu of Long Short Term Memory (LSTM) [228], is because its structure is simpler and its efficiency is higher. The additional time and energy overhead of GR compared to BackboneNet is between 12% to 89%. It is still within our acceptable range, but suggests that an avenue of future work might be investigating efficiency issues in the feature reconstruction module. Regarding the memory usage, the relationship seems to be more unpredictable. We find the BackboneNet has the highest memory usage in both RealWorld-HAR and EEG-MMID dataset. Also, larger memory usage is caused by DeepSense compared to CNN, though it has much fewer FLOPs, as shown in Table 6.2. In general, GraphRecovery does not pose a significant memory overhead compared to the BackboneNet. We will continue to investigate the low-level implementations about why the BackboneNet can have a higher memory usage.

6.7 RELATED WORK

Missing sensors is a prevalent problem in multi-sensor IoT systems and wireless sensor networks. Hossain *et al.* [229] have empirically investigated the impact of randomly missing sensing values on extracted features and classification results for conventional machine

learning models, which are based on feature engineering. The handling of missing sensor values has been studied for a long time. The earliest work focused on recovering missing sensor data directly at the signal level [216, 217, 230, 231, 232, 233]. Most of these approaches use data imputation methods that rely on time locality and space locality to fill-in the missing sensor readings. For example, Yi *et al.* [216] use a hybrid autoregressive method to fill missing sensor values, considering both temporal correlations and spatial correlations. Huang *et al.* [233] rely on non-negative matrix factorization (NMF) to impute the missing values. Imani *et al.* [232] utilize a complicated Gaussian process for high-dimensional data imputation. However, most of these approaches can't effectively handle the case where data from the same sensors are completely missing for a long time, because they impute each sensor reading independently, thus the assembled sequence can not preserve the semantic information contained in the original sensing sequence. Even if some autoregressive models [216] can recover the signal sequence as a whole, they can not be extended to handle randomly missing sensors. On the contrary, they have to train an individual model for each missing sensor situation, and assume all the remaining sensors are available.

The emergence and prevalence of deep learning has revolutionized data processing in IoT systems. Feature engineering is replaced by automatic representation learning by neural networks, so that learning is directly performed on sensing signals. Regarding the dynamic sensor configuration in sensing models, Rey *et al.* [234] proposed a similarity-based approach to integrate new sensors into an existing recognition system in a semi-supervised manner. Meanwhile, several neural network based general missing sensor handling algorithms (*i.e.*, algorithms where one model handles all missing sensor situations) have been proposed [143, 199, 201, 202, 203]. Most papers [201, 202, 203] borrow the idea from denoising autoencoders [204]. During training, they randomly drop some sensors before the input layer of the encoder, and let the autoencoder automatically learn to reconstruct data for missing sensors. However, there is no control on the information flow in these algorithms. Thus, useful information from available sensors and null information from missing sensors are mixed together at the latent embedding layer. The result is that the data of available sensors is polluted after the encoder-decoder pipeline. The remaining papers [143, 199] try to automatically adjust the model according to the missing sensors. Vaizman *et al.* [199] simulate the sensor missing situations by randomly removing some sensors during the training. However, incompatible information under different sensor combinations (*e.g.*, upper body movement vs. whole body movement), especially when a large number of sensors is involved, may cause extra learning difficulties to the model. Liu *et al.* [143] design an attention mechanism when conducting sensor fusion, so that the missing sensors can be automatically ignored by the attention module. Since we already know which sensors are missing in each

data sample, it is apparently easier and more effective to utilize a data-dependent mask to push the neural network to only extract information from available sensors. Besides, information incompatibility problems still exist in their design. In our graph recovery design, we tackle the information incompatibility problem directly by reconstructing features for missing sensors based on their spatial connections and learned latent sensor correlations. The information flow is controlled by the corresponding gating mechanism, and output substitution is performed to prevent feature pollution at available sensors.

CHAPTER 7: CONCLUSION & FUTURE WORK

In this dissertation, we have explored how to apply the *attention-based design philosophy* to optimize the *efficiency* and *efficacy* of machine perception in intelligent cyber-physical systems, where DNNs are mainly used as the recognition model to extract knowledge from the sensing data.

- From the **efficiency** perspective, inspired by the observed *priority inversion* issue in current machine perception pipelines, we proposed to process the input data (*e.g.*, images) using a criticality-aware manner at finer-grained granularities. For instance, in a vision perception system, we first sliced the input images into semantically meaningful partial regions, either utilizing cross-cueing from an external sensor or self-cueing based on temporal correlations in video streams, and then a set of criticality-based batch-aware real-time scheduling algorithms were proposed to schedule the processing of sliced partial regions. Different criticality designs, like physical distance/relative velocity-based criticality, uncertainty-based criticality, have been investigated and evaluated. After extending the work to a distributed multi-camera perception system, we found additional efficiency saving can be achieved by exploiting the spatial correlations among cameras.

- From the **efficacy** perspective, we mainly focused on enhancing the robustness of DNNs to handle the imperfections of sensing data in practical deployment. On one hand, we designed a novel global attention mechanism for multi-sensor information fusion, where the heterogeneous data quality and information relevance to the task at different sensor types and locations were captured and addressed. On the other hand, we built a GNN-based feature reconstruction mechanism for handling missing sensors in a distributed sensing system, where the spatial data correlations among physically connected sensors were exploited, such that a single recognition model can remain effective under different sets of available sensors, without requiring any retraining.

We successfully demonstrated the success of attention-based design philosophy in our past explorations. To summarize, it essentially represents the ideas of: 1) selectively scheduling the invocation of computation-intensive neural network models to achieve real-time inference on resource-limited embedded platforms, and 2) dynamically adjusting the weighting mechanism among heterogeneous input within neural networks to enhance the model robustness against data noises and failures. Looking forward to the future, we will continue to study the following directions in achieving the attention-based machine perception.

- **Generalizing the criticality design in attention scheduling.** While we have discussed several ways of instantiating the criticality design in attention-based scheduling, there is a much broader space remaining for future explorations. In the external-cueing framework, besides the distance-based and velocity-based criticality, we can take more semantic information into consideration. For example, a further object in the same lane can be more critical than a close object in different lanes. Besides, the prioritization in the perception system should appropriately integrate the traffic regulations because the safety of autonomous driving is not only about collision avoidance, but also about obeying the traffic laws, *e.g.*, properly reacting to traffic signs and traffic light information. The (location) uncertainty-based prioritization mechanism can be extended to a general notion of change. Changes that occur in an unpredicted way should receive more attention. For example, if an object appears where it was not predicted, then something unexpected has occurred that requires attention. Information gain comes from the degree of surprise on the perceived scene. Therefore, it would be interesting to develop a surprised/entropy-based criticality for attention scheduling in machine perception. Furthermore, a hybrid scheme, integrating multiple criticality policies, might be better at dealing with complicated sensing scenarios. How to effectively combine multiple policies in a context-driven fashion, properly adjust their composition, and resolve their conflicts is a very promising but challenging research problem.

- **Integrating neural attention with human-designed attention.** Our works so far mainly use human-designed attention to control the resource allocation in neural network execution. However, neural attention mechanism is also extensively applied to optimize the neural network accuracy by focusing on the important parts of the input. How to effectively combine the two approaches in terms of both efficiency and efficacy optimization in one policy will be an interesting direction to explore. On one hand, the neural attention mechanism is better at understanding the neural network performance internally; on the other hand, the human-designed attention definitions are better at communicating with the physical world information (*e.g.*, distance, velocity, and traffic laws). We anticipate that the main challenge will be injecting the human input about the physical world into the automatic execution pipeline of neural networks with extra supervision or regulation, and resolving the potential conflicts between the two types of attention.

- **Interactive scheduling of machine perception.** We did not consider the manipulation or control on the sensor actions in our past works. However, the scheduling

process itself can be interactive, where we can not only selectively process parts of the perceived information, but also proactively control the sensors to look at regions of more interest. For instance, when using Pan-Tilt-Zoom (PTZ) cameras for machine perception, we can control the direction and zoom distance of the camera to dynamically adjust the range of their perception. Alternatively, when using mobile sensors/cameras for perception, we can control the movement patterns of sensor nodes to achieve high-quality perception of a large target region with only a few sensor nodes. However, it is still a challenging problem to formulate and solve, because we have to consider the sensor operation latency (*i.e.*, zoom, move from one point to another) in the scheduling framework, to make sure the sensors operate in an attainable, smooth, and information-maximizing mode.

- **Optimizing distributed machine perception under network dynamics.** We made an initial attempt to generalize the attention-based real-time scheduling to distributed perception systems, but there are much more we can do. For instance, in a large scale distributed perception system, due to the extremely limited onboard processing capacity on sensors, the sensing data is typically offloaded to a cloud/edge server for processing. In this scenario, sensors are mostly wirelessly connected to the Internet, such that the network bandwidth becomes the bottleneck of the whole computation pipeline. How to minimize the amount of data to be transmitted, without degrading the recognition model performance, is an interesting problem where we can apply the attention-based design philosophy. By doing so, the system scalability could be significantly improved, under the same network constraint. In addition, how to dynamically adjust the amount of data to be transmitted at each sensor, in response to the dynamic network bandwidth, to minimize the end-to-end response latency, is another interesting problem to investigate.

- **Integrating physical domain knowledge into neural perception models.** Although DNNs have outperformed conventional physical models or manual-feature-based machine learning models in many recognition tasks, there is still plenty of room where the physical domain knowledge could help enhance the DNN models. The poor explainability, high dependency on the training data quality, and high sensitivity of the DNN models are the main factors that restrict their deployment in practical industrial environment, since their behaviors can be randomly bad at the outliers of their training data distributions. Under these cases, physical domain knowledge can be used as a special form of supervision, regularization, or augmentation, to teach the neural networks to avoid obvious prediction errors that humans can easily distinguish. The

reliability, extendibility, and explainability of DNNs are anticipated to be significantly enhanced. The challenge mainly lies in how to integrate the knowledge of physical models and neural models in a rational way, and resolve their possible conflicts, such that the integrated model can achieve superior performance compared to any single model.

Attention-based machine perception is generally a promising direction, and we are excited about its future development and evolvement. We hope the works reported in this dissertation, can become a starting point that inspires more explorations from the research community on optimizing the usage of DNN-based machine perception models, in terms of both efficiency and efficacy, during their practical deployment, towards building the next-generation intelligent cyber-physical systems.

REFERENCES

[1] M. I. Posner, C. R. Snyder, and R. Solso, "Attention and cognitive control," *Cognitive psychology: Key readings*, vol. 205, 2004.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[3] 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.

[4] 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.

[5] "Driverless guru," https://www.driverlessguru.com/self-driving-cars-facts-and-figures, 2020.

[6] D. Bamburry, "Drones: Designed for product delivery," *Design Management Review*, vol. 26, no. 1, pp. 40–48, 2015.

[7] T. Abdelzaher, N. Ayanian, T. Basar, S. Diggavi, J. Diesner, D. Ganesan, R. Govindan, S. Jha, T. Lepoint, B. Marlin et al., "Toward an internet of battlefield things: A resilience perspective," *Computer*, vol. 51, no. 11, pp. 24–36, 2018.

[8] D. Feil-Seifer and M. J. Matarić, "Socially assistive robotics," *IEEE Robotics & Automation Magazine*, vol. 18, no. 1, pp. 24–31, 2011.

[9] T. P. Baker and A. Shaw, "The cyclic executive model and ada," *Real-Time Systems*, vol. 1, no. 1, pp. 7–25, 1989.

[10] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *RTSS*, vol. 89, 1989, pp. 166–171.

[11] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in neural information processing systems*, 2015, pp. 577–585.

[12] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang, "Bottom-up and top-down attention for image captioning and visual question answering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6077–6086.

[13] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[14] Y. Wang, M. Huang, L. Zhao et al., "Attention-based lstm for aspect-level sentiment classification," in *Proceedings of the 2016 conference on empirical methods in natural language processing*, 2016, pp. 606–615.

[15] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3156–3164.

[16] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, 2015, pp. 2048–2057.

[17] M. M. Minderhoud and P. H. Bovy, "Extended time-to-collision measures for road traffic safety assessment," *Accident Analysis & Prevention*, vol. 33, no. 1, pp. 89–97, 2001.

[18] I. Bogoslavskyi and C. Stachniss, "Fast range image-based segmentation of sparse 3d laser scans for online operation," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 163–169.

[19] 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.

[20] J. W. 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.

[21] 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," in *Foundations of Real-Time Computing: Scheduling and Resource Management*. Springer, 1991, pp. 203–249.

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

[23] S. Yao, Y. Hao, Y. Zhao, A. Piao, H. Shao, D. Liu, S. Liu, S. Hu, D. Weerakoon, K. Jayarajah et al., "Eugene: Towards deep intelligence as a service," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1630–1640.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[25] S. Yao, Y. Zhao, H. Shao, A. Zhang, C. Zhang, S. Li, and T. Abdelzaher, "Rdeepsense: Reliable deep mobile computing models with uncertainty estimations," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, pp. 1–26, 2018.

[26] M. Himmelsbach, F. V. Hundelshausen, and H.-J. Wuensche, "Fast segmentation of 3d point clouds for ground vehicles," in *2010 IEEE Intelligent Vehicles Symposium*. IEEE, 2010, pp. 560–565.

[27] 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.

[28] B. Hajek, "On the competitiveness of on-line scheduling of unit-length packets with hard deadlines in slotted time," in *Proceedings of the 2001 Conference on Information Sciences and Systems*, 2001.

[29] 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.

[30] R. Pujol, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. J. Cazorla, "Generating and exploiting deep learning variants to increase heterogeneous resource utilization in the nvidia xavier," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, vol. 23, 2019.

[31] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine et al., "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2446–2454.

[32] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[33] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE international conference on image processing (ICIP)*. IEEE, 2016, pp. 3464–3468.

[34] J. Wang, P. He, and W. Coo, "Study on the hungarian algorithm for the maximum likelihood data association problem," *Journal of Systems Engineering and Electronics*, vol. 18, no. 1, pp. 27–32, 2007.

[35] A. Torralba, "How many pixels make an image?" *Visual neuroscience*, vol. 26, no. 1, pp. 123–131, 2009.

[36] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[37] Y. Hu, S. Liu, T. Abdelzaher, M. Wigness, and P. David, "On exploring image resizing for optimizing criticality-based machine perception," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2021.

[38] S. Liu, S. Yao, X. Fu, R. Tabish, S. Yu, H. Yun, L. Sha, and T. Abdelzaher, "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *In Proc. IEEE Real-time Systems Symposium (RTSS)*, December 2020.

[39] S. Yao, Y. Zhao, A. Zhang, S. Hu, H. Shao, C. Zhang, L. Su, and T. Abdelzaher, "Deep learning for the internet of things," *Computer*, vol. 51, no. 5, pp. 32–41, 2018.

[40] S. Liu, S. Yao, J. Li, D. Liu, T. Wang, H. Shao, and T. Abdelzaher, "Giobalfusion: A global attentional deep learning framework for multisensor information fusion," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 1, pp. 1–27, 2020.

[41] S. Liu, S. Yao, Y. Huang, D. Liu, H. Shao, Y. Zhao, J. Li, T. Wang, R. Wang, C. Yang et al., "Handling missing sensors in topology-aware iot applications with gated graph neural network," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 3, pp. 1–31, 2020.

[42] T. Abdelzaher, Y. Hao, K. Jayarajah, A. Misra, P. Skarin, S. Yao, D. Weerakoon, and K.-E. Årzén, "Five challenges in cloud-enabled intelligence and control," *ACM Transactions on Internet Technology (TOIT)*, vol. 20, no. 1, pp. 1–19, 2020.

[43] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 2, pp. 423–443, 2018.

[44] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[45] S. Yao, A. Piao, W. Jiang, Y. Zhao, H. Shao, S. Liu, D. Liu, J. Li, T. Wang, S. Hu et al., "Stfnets: Learning sensing signals from the time-frequency perspective with short-time fourier neural networks," pp. 2192–2202, 2019.

[46] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018.

[47] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[48] 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.

[49] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[50] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[51] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 2017, p. 4.

[52] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems*, 2014, pp. 1269–1277.

[53] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. ACM, 2016, pp. 176–189.

[54] B. Minnehan and A. Savakis, "Cascaded projection: End-to-end network compression and acceleration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 715–10 724.

[55] Y. Wang, C. Xu, S. You, D. Tao, and C. Xu, "Cnnpack: packing convolutional neural networks in the frequency domain," in *Advances in Neural Information Processing Systems*, 2016, pp. 253–261.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

[60] M. H. Santriaji and H. Hoffmann, "Merlot: Architectural support for energy-efficient real-time processing in gpus," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 214–226.

[61] 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.

[62] R. Cavicchioli, N. Capodieci, M. Solieri, and M. Bertogna, "Novel methodologies for predictable cpu-to-gpu command offloading," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[63] S. Bateni, H. Zhou, Y. Zhu, and C. Liu, "Predjoule: A timing-predictable energy optimization framework for deep neural networks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 107–118.

[64] S. Bateni and C. Liu, "Predictable data-driven resource management: an implementation using autoware on autonomous platforms," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 339–352.

[65] 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.

[66] M. Khayatian, M. Mehrabian, and A. Shrivastava, "Rim: Robust intersection management for connected autonomous vehicles," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 35–44.

[67] D. Zhang, N. Vance, Y. Zhang, M. T. Rashid, and D. Wang, "Edgebatch: Towards ai-empowered optimal task batching in intelligent edge systems," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 366–379.

[68] J. Shin, Y. Baek, and S. H. Son, "Fundamental topology-based routing protocols for autonomous vehicles," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2016, pp. 265–265.

[69] L. Li, H. Xiong, Z. Guo, J. Wang, and C.-Z. Xu, "Smartpc: Hierarchical pace control in real-time federated learning system," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 406–418.

[70] S. Aoki and R. R. Rajkumar, "A configurable synchronous intersection protocol for self-driving vehicles," in *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2017, pp. 1–11.

[71] Y. Ikeda, Y. Yanagisawa, Y. Kishino, S. Mizutani, Y. Shirai, T. Suyama, K. Matsumura, and H. Noma, "Reduction of communication cost for edge-heavy sensor using divided cnn," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 244–245.

[72] S. Aoki and R. Rajkumar, "V2v-based synchronous intersection protocols for mixed traffic of human-driven and self-driving vehicles," in *2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2019, pp. 1–11.

[73] S. K. Kwon, E. Hyun, J.-H. Lee, J. Lee, and S. H. Son, "A low-complexity scheme for partially occluded pedestrian detection using lidar-radar sensor fusion," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2016, pp. 104–104.

[74] S. Li, D. Liu, C. Xiang, J. Liu, Y. Ling, T. Liao, and L. Liang, "Fitcnn: A cloud-assisted lightweight convolutional neural network framework for mobile devices," in *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2017, pp. 1–6.

[75] M. G. Bechtel, E. McEllhiney, M. Kim, and H. Yun, "Deeppicar: A low-cost deep neural network-based autonomous car," pp. 11–21, 2018.

[76] K. Mikami, Y. Chen, J. Nakazawa, Y. Iida, Y. Kishimoto, and Y. Oya, "Deepcounter: Using deep learning to count garbage bags," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 1–10.

[77] M.-H. Cheng, Q. Sun, and C.-H. Tu, "An adaptive computation framework of distributed deep learning models for internet-of-things applications," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 85–91.

[78] T. Nukita, Y. Kishimoto, Y. Iida, M. Kawano, T. Yonezawa, and J. Nakazawa, "Damaged lane markings detection method with label propagation," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 203–208.

[79] M. Bojarski et al., "End-to-End Learning for Self-Driving Cars," 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[80] N. Otterness, M. Yang, S. Rust, E. Park, J. H. Anderson, F. D. Smith, A. Berg, and S. Wang, "An evaluation of the nvidia tx1 for supporting real-time computer-vision workloads," in *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2017, pp. 353–364.

[81] M. Yang, N. Otterness, T. Amert, J. Bakita, J. H. Anderson, and F. D. Smith, "Avoiding pitfalls when using nvidia gpus for real-time tasks in autonomous systems," in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[82] J. Park, J.-H. Lee, and S. H. Son, "A survey of obstacle detection using vision sensor for autonomous vehicles," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2016, pp. 264–264.

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

[84] W. Kang and J. Chung, "Deeprt: predictable deep learning inference for cyber-physical systems," *Real-Time Systems*, vol. 55, no. 1, pp. 106–135, 2019.

[85] 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)*. IEEE, 2020, pp. 15–29.

[86] 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)*. IEEE, 2020, pp. 174–187.

[87] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," pp. 779–788, 2016.

[88] M. Daily, S. Medasani, R. Behringer, and M. Trivedi, "Self-driving cars," *Computer*, vol. 50, no. 12, pp. 18–23, 2017.

[89] J. Scott and C. Scott, "Drone delivery models for healthcare," in *Proceedings of the 50th Hawaii international conference on system sciences*, 2017.

[90] W.-Y. G. Louie, T. Vaquero, G. Nejat, and J. C. Beck, "An autonomous assistive robot for planning, scheduling and facilitating multi-user activities," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 5292–5298.

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

[92] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[93] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[94] Z. Huang, Z. Chen, Q. Li, H. Zhang, and N. Wang, "1st place solutions of waymo open dataset challenge 2020–2d object detection track," *arXiv preprint arXiv:2008.01365*, 2020.

[95] J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon, "Clio: enabling automatic compilation of deep learning pipelines across iot and cloud," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–12.

[96] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys)*, 2020.

[97] T. Kroeger, R. Timofte, D. Dai, and L. Van Gool, "Fast optical flow using dense inverse search," in *European Conference on Computer Vision*. Springer, 2016, pp. 471–488.

[98] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE transactions on Image Processing*, vol. 9, no. 2, pp. 287–290, 2000.

[99] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2758–2766.

[100] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel, "The pinwheel: A real-time scheduling problem," in *Proceedings of the 22nd Hawaii International Conference of System Science*, 1989, pp. 693–702.

[101] C. Grana, D. Borghesani, and R. Cucchiara, "Optimized block-based connected components labeling with decision trees," *IEEE Transactions on Image Processing*, vol. 19, no. 6, pp. 1596–1609, 2010.

[102] T.-W. Chin, R. Ding, and D. Marculescu, "Adascale: Towards real-time video object detection using adaptive scaling," in *Systems and Machine Learning Conference*, 2019.

[103] R. Xu, C.-l. Zhang, P. Wang, J. Lee, S. Mitra, S. Chaterji, Y. Li, and S. Bagchi, "Approxdet: content and contention-aware approximate object detection for mobiles," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 449–462.

[104] S. Lee and S. Nirjon, "Fast and scalable in-memory deep multitask learning via neural weight virtualization," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 175–190.

[105] 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 *In Proc. IEEE Real-time Systems Symposium (RTSS)*, December 2020.

[106] 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.

[107] 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*, 2021.

[108] X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei, "Deep feature flow for video recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2349–2358.

[109] S. Wang, H. Lu, and Z. Deng, "Fast object detection in compressed video," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7104–7113.

[110] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei, "Flow-guided feature aggregation for video object detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 408–417.

[111] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, "Deepcache: Principled cache for mobile deep vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 129–144.

[112] M. Buckler, P. Bedoukian, S. Jayasuriya, and A. Sampson, "Eva^2: Exploiting temporal redundancy in live computer vision," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 533–546.

[113] L. Cavigelli, P. Degen, and L. Benini, "Cbinfer: Change-based inference for convolutional neural networks on video data," in *Proceedings of the 11th International Conference on Distributed Smart Cameras*, 2017, pp. 1–8.

[114] S. Zhang, W. Lin, P. Lu, W. Li, and S. Deng, "Kill two birds with one stone: Boosting both object detection accuracy and speed with adaptive patch-of-interest composition," in *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE, 2017, pp. 447–452.

[115] Z. Song, B. Fu, F. Wu, Z. Jiang, L. Jiang, N. Jing, and X. Liang, "Drq: dynamic region-based quantization for deep neural network acceleration," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 1010–1021.

[116] A. R. Kumar, B. Ravindran, and A. Raghunathan, "Pack and detect: Fast object detection in videos using region-of-interest packing," in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, 2019, pp. 150–156.

[117] H. Mao, T. Kong, and W. J. Dally, "Catdet: Cascaded tracked detector for efficient object detection from video," *arXiv preprint arXiv:1810.00434*, 2018.

[118] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live Video Analytics at Scale with Approximation and Delay-Tolerance," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, ser. NSDI '17, 2017.

[119] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 359–376.

[120] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 557–570.

[121] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: optimizing neural network queries over video at scale," *arXiv preprint arXiv:1703.02529*, 2017.

[122] S. Jain, X. Zhang, Y. Zhou, G. Ananthanarayanan, J. Jiang, Y. Shu, P. Bahl, and J. Gonzalez, "Spatula: Efficient cross-camera video analytics on large camera networks," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 110–124.

[123] S. Jain, G. Ananthanarayanan, J. Jiang, Y. Shu, and J. Gonzalez, "Scaling video analytics systems to large camera deployments," in *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, 2019, pp. 9–14.

[124] M. Naphade, S. Wang, D. C. Anastasiu, Z. Tang, M.-C. Chang, X. Yang, Y. Yao, L. Zheng, P. Chakraborty, C. E. Lopez, A. Sharma, Q. Feng, V. Ablavsky, and S. Sclaroff, "The 5th ai city challenge," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2021.

[125] Z. Tang, M. Naphade, M.-Y. Liu, X. Yang, S. Birchfield, S. Wang, R. Kumar, D. Anastasiu, and J.-N. Hwang, "Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019, p. 8797–8806.

[126] M. Andriluka, S. Roth, and B. Schiele, "People-tracking-by-detection and people-detection-by-tracking," in *2008 IEEE Conference on computer vision and pattern recognition*. IEEE, 2008, pp. 1–8.

[127] Z. Zhou, D. Yin, J. Ding, Y. Luo, M. Yuan, and C. Zhu, "Collaborative tracking method in multi-camera system," *Journal of Shanghai Jiaotong University (Science)*, vol. 25, pp. 802–810, 2020.

[128] M. R. Garey and D. S. Johnson, *Computers and intractability*. freeman San Francisco, 1979, vol. 174.

[129] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 253–266.

[130] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h. 264/avc video coding standard," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560–576, 2003.

[131] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu, "Focus: Querying large video datasets with low latency and low cost," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 269–286.

[132] Y. Zhang and A. Kumar, "Panorama: a data system for unbounded vocabulary querying over video," *Proceedings of the VLDB Endowment*, vol. 13, no. 4, pp. 477–491, 2019.

[133] M. R. Anderson, M. Cafarella, G. Ros, and T. F. Wenisch, "Physical representation-based predicate optimization for a visual analytics database," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1466–1477.

[134] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor, "Scaling video analytics on constrained edge nodes," *arXiv preprint arXiv:1905.13536*, 2019.

[135] H. Guo, S. Yao, Z. Yang, Q. Zhou, and K. Nahrstedt, "Crossroi: Cross-camera region of interest optimization for efficient real time video analytics at scale," *arXiv preprint arXiv:2105.06524*, 2021.

[136] S. Paul, U. Drolia, Y. C. Hu, and S. T. Chakradhar, "Aqua: Analytical quality assessment for optimizing video analytics systems," *arXiv preprint arXiv:2101.09752*, 2021.

[137] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 115–131.

[138] X. Zeng, B. Fang, H. Shen, and M. Zhang, "Distream: scaling live video analytics with workload-adaptive distributed edge intelligence," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 409–421.

[139] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1421–1429.

[140] W. Xu, Y. Shen, N. Bergmann, and W. Hu, "Sensor-assisted face recognition system on smart glass via multi-view sparse representation classification," in *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2016, pp. 1–12.

[141] S. A. Rokni and H. Ghasemzadeh, "Synchronous dynamic view learning: a framework for autonomous training of activity recognition models using wearable sensors," in *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks*. ACM, 2017, pp. 79–90.

[142] S. Shen, H. Wang, and R. Roy Choudhury, "I am a smartwatch and i can track my user's arm," in *Proceedings of the 14th annual international conference on Mobile systems, applications, and services*. ACM, 2016, pp. 85–96.

[143] Y. Liu, Z. Li, Z. Liu, and K. Wu, "Real-time arm skeleton tracking and gesture inference tolerant to missing wearable sensors," in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2019, pp. 287–299.

[144] X. Liu, P. Ghosh, O. Ulutan, B. Manjunath, K. Chan, and R. Govindan, "Caesar: cross-camera complex activity recognition," in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. ACM, 2019, pp. 232–244.

[145] Y. Yuan, G. Xun, K. Jia, and A. Zhang, "A multi-view deep learning method for epileptic seizure detection using short-time fourier transform," in *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. ACM, 2017, pp. 213–222.

[146] Y. Zhao, S. Yao, D. Liu, H. Shao, and S. Liu, "Greenroute: A generalizable fuel-saving vehicular navigation service," in *2019 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2019, pp. 1–10.

[147] H. Xue, W. Jiang, C. Miao, Y. Yuan, F. Ma, X. Ma, Y. Wang, S. Yao, W. Xu, A. Zhang et al., "Deepfusion: A deep learning framework for the fusion of heterogeneous sensory data," in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2019, pp. 151–160.

[148] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "Deepsense: A unified deep learning framework for time-series mobile sensing data processing," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 351–360.

[149] V. Radu, N. D. Lane, S. Bhattacharya, C. Mascolo, M. K. Marina, and F. Kawsar, "Towards multimodal deep learning for activity recognition on mobile devices," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*. ACM, 2016, pp. 185–188.

[150] X. Wang, X. Wang, and S. Mao, "Rf sensing in the internet of things: A general deep learning framework," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 62–67, 2018.

[151] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE Press, 2016, p. 23.

[152] N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, and F. Kawsar, "Squeezing deep learning into mobile and embedded devices," *IEEE Pervasive Computing*, vol. 16, no. 3, pp. 82–88, 2017.

[153] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. ACM, 2016, pp. 176–189.

[154] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in neural information processing systems*, 2014, pp. 3320–3328.

[155] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *2012 16th International Symposium on Wearable Computers*. IEEE, 2012, pp. 108–109.

[156] T. Sztyler and H. Stuckenschmidt, "On-body localization of wearable devices: An investigation of position-aware activity recognition," in *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2016, pp. 1–9.

[157] K. Altun, B. Barshan, and O. Tunçel, "Comparative study on classifying human activities with miniature inertial and magnetic sensors," *Pattern Recognition*, vol. 43, no. 10, pp. 3605–3620, 2010.

[158] M. Bachlin, D. Roggen, G. Troster, M. Plotnik, N. Inbar, I. Meidan, T. Herman, M. Brozgol, E. Shaviv, N. Giladi et al., "Potentials of enhanced context awareness in wearable assistants for parkinson's disease patients with the freezing of gait syndrome," in *2009 International Symposium on Wearable Computers*. IEEE, 2009, pp. 123–130.

[159] S. Yao, Y. Zhao, H. Shao, D. Liu, S. Liu, Y. Hao, A. Piao, S. Hu, S. Lu, and T. F. Abdelzaher, "Sadeepsense: Self-attention deep learning framework for heterogeneous on-device sensors in internet of things applications," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1243–1251.

[160] C. Wang, M. Peng, T. A. Olugbade, N. D. Lane, A. C. D. C. Williams, and N. Bianchi-Berthouze, "Learning bodily and temporal attention in protective movement behavior detection," *arXiv preprint arXiv:1904.10824*, 2019.

[161] M. Zeng, H. Gao, T. Yu, O. J. Mengshoel, H. Langseth, I. Lane, and X. Liu, "Understanding and improving recurrent networks for human activity recognition by continuous attention," in *Proceedings of the 2018 ACM International Symposium on Wearable Computers*. ACM, 2018, pp. 56–63.

[162] B. Chandrasekaran, S. Gangadhar, and J. M. Conrad, "A survey of multisensor fusion techniques, architectures and methodologies," in *SoutheastCon 2017*. IEEE, 2017, pp. 1–8.

[163] O. Rippel, J. Snoek, and R. P. Adams, "Spectral representations for convolutional neural networks," in *Advances in neural information processing systems*, 2015, pp. 2449–2457.

[164] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[165] S. Jetley, N. A. Lord, N. Lee, and P. H. Torr, "Learn to pay attention," *arXiv preprint arXiv:1804.02391*, 2018.

[166] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[167] "Tensorflow lite interpreter." [Online]. Available: https://www.tensorflow.org/lite/guide/inference

[168] "Monsoon high voltage power monitor." [Online]. Available: https://www.msoon.com/online-store/High-Voltage-Power-Monitor-Part-Number-AAA10F-p90002590

[169] D. Hall and J. Llinas, *Multisensor data fusion*. CRC press, 2001.

[170] N. Neverova, C. Wolf, G. Taylor, and F. Nebout, "Moddrop: adaptive multi-modal gesture recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 8, pp. 1692–1706, 2015.

[171] V. Vielzeuf, A. Lechervy, S. Pateux, and F. Jurie, "Multi-level sensor fusion with deep learning," *CoRR*, vol. abs/1811.02447, 2018. [Online]. Available: http://arxiv.org/abs/1811.02447

[172] Z. Liu, W. Zhang, T. Q. Quek, and S. Lin, "Deep fusion of heterogeneous sensor data," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 5965–5969.

[173] V. Radu, C. Tong, S. Bhattacharya, N. D. Lane, C. Mascolo, M. K. Marina, and F. Kawsar, "Multimodal deep learning for activity and context recognition," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, p. 157, 2018.

[174] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[175] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[176] S. Chaudhari, G. Polatkan, R. Ramanath, and V. Mithal, "An attentive survey of attention models," *arXiv preprint arXiv:1904.02874*, 2019.

[177] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[178] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7794–7803.

[179] Y. Chen, Y. Kalantidis, J. Li, S. Yan, and J. Feng, "Aˆ 2-nets: Double attention networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 352–361.

[180] "Amazon alexa." [Online]. Available: https://developer.amazon.com/en-US/alexa

[181] "Google assistant." [Online]. Available: https://assistant.google.com/

[182] M. Manisha, K. Neeraja, V. Sindhura, and P. Ramaya, "Iot on heart attack detection and heart rate monitoring," *International Journal of Innovation in Engineering and Technology (IJIET)*, 2016.

[183] N. Constant, O. Douglas-Prawl, S. Johnson, and K. Mankodiya, "Pulse-glasses: An unobtrusive, wearable hr monitor with internet-of-things functionality," in *2015 IEEE 12th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*. IEEE, 2015, pp. 1–5.

[184] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[185] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.

[186] Y. Jie, J. Y. Pei, L. Jun, G. Yun, and X. Wei, "Smart home system based on iot technologies," in *2013 International Conference on Computational and Information Sciences*. IEEE, 2013, pp. 1789–1791.

[187] "Google nest." [Online]. Available: https://nest.com/

[188] J. Sherly and D. Somasundareswari, "Internet of things based smart transportation systems," *International Research Journal of Engineering and Technology*, vol. 2, no. 7, pp. 1207–1210, 2015.

[189] A. Roy, J. Siddiquee, A. Datta, P. Poddar, G. Ganguly, and A. Bhattacharjee, "Smart traffic & parking management using iot," in *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2016, pp. 1–3.

[190] Y. Zhao, S. Yao, D. Liu, H. Shao, S. Liu, and T. Abdelzaher, "Simulation evaluation of fuel-saving systems in the city of chicago," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019, pp. 1–9.

[191] D. Vasisht, Z. Kapetanovic, J. Won, X. Jin, R. Chandra, S. Sinha, A. Kapoor, M. Sudarshan, and S. Stratman, "Farmbeats: An iot platform for data-driven agriculture," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 515–529.

[192] S. Prathibha, A. Hongal, and M. Jyothi, "Iot based monitoring system in smart agriculture," in *2017 international conference on recent advances in electronics and communication technology (ICRAECT)*. IEEE, 2017, pp. 81–84.

[193] P. Verma and S. K. Sood, "Fog assisted-iot enabled patient health monitoring in smart homes," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1789–1796, 2018.

[194] D. O. Bos et al., "Eeg-based emotion recognition," *The Influence of Visual and Auditory Stimuli*, vol. 56, no. 3, pp. 1–17, 2006.

[195] S. Yao, Y. Zhao, H. Shao, C. Zhang, A. Zhang, S. Hu, D. Liu, S. Liu, L. Su, and T. Abdelzaher, "Sensegan: Enabling deep learning for internet of things with a semi-supervised framework," vol. 2, no. 3. ACM New York, NY, USA, 2018, pp. 1–21.

[196] S. Yao, Y. Zhao, H. Shao, C. Zhang, A. Zhang, D. Liu, S. Liu, L. Su, and T. Abdelzaher, "Apdeepsense: Deep learning uncertainty estimation without the pain for iot applications," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 334–343.

[197] M. K. Markey, G. D. Tourassi, M. Margolis, and D. M. DeLong, "Impact of missing data in evaluating artificial neural networks trained on complete data," *Computers in Biology and Medicine*, vol. 36, no. 5, pp. 516–525, 2006.

[198] R. Qiao, C. Qing, T. Zhang, X. Xing, and X. Xu, "A novel deep-learning based framework for multi-subject emotion recognition," in *2017 4th International Conference on Information, Cybernetics and Computational Social Systems (ICCSS)*. IEEE, 2017, pp. 181–185.

[199] Y. Vaizman, N. Weibel, and G. Lanckriet, "Context recognition in-the-wild: Unified model for multi-modal sensors and multi-label classification," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, pp. 1–22, 2018.

[200] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[201] N. Jaques, S. Taylor, A. Sano, and R. Picard, "Multimodal autoencoder: A deep learning approach to filling in missing sensor data and enabling better mood prediction," in *2017 Seventh International Conference on Affective Computing and Intelligent Interaction (ACII)*. IEEE, 2017, pp. 202–208.

[202] A. Saeed, T. Ozcelebi, and J. Lukkien, "Synthesizing and reconstructing missing sensory modalities in behavioral context recognition," *Sensors*, vol. 18, no. 9, p. 2967, 2018.

[203] C. Hong, J. Yu, J. Wan, D. Tao, and M. Wang, "Multimodal deep autoencoder for human pose recovery," *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5659–5670, 2015.

[204] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

[205] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[206] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[207] O. Banos, M. A. Toth, M. Damas, H. Pomares, and I. Rojas, "Dealing with the effects of sensor displacement in wearable activity recognition," *Sensors*, vol. 14, no. 6, pp. 9995–10 023, 2014.

[208] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw, "Bci2000: a general-purpose brain-computer interface (bci) system," *IEEE Transactions on biomedical engineering*, vol. 51, no. 6, pp. 1034–1043, 2004.

[209] M. K. Gill, T. Asefa, Y. Kaheil, and M. McKee, "Effect of missing data on performance of learning algorithms for hydrologic predictions: Implications to an imputation technique," *Water resources research*, vol. 43, no. 7, 2007.

[210] C. M. Ennett, M. Frize, and C. R. Walker, "Influence of missing values on artificial neural network performance," in *Medinfo*, 2001, pp. 449–453.

[211] O. Baños, M. Damas, H. Pomares, I. Rojas, M. A. Tóth, and O. Amft, "A benchmark dataset to evaluate sensor displacement in activity recognition," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, 2012, pp. 1026–1035.

[212] V. Jurcak, D. Tsuzuki, and I. Dan, "10/20, 10/10, and 10/5 systems revisited: their validity as relative head-surface-based positioning systems," *Neuroimage*, vol. 34, no. 4, pp. 1600–1611, 2007.

[213] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," *Computational Social Networks*, vol. 6, no. 1, p. 11, 2019.

[214] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[215] "Electroencephalography." [Online]. Available: https://en.wikipedia.org/wiki/Electroencephalography

[216] X. Yi, Y. Zheng, J. Zhang, and T. Li, "St-mvl: filling missing values in geo-sensory time series data," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 2016, pp. 2704–2710.

[217] L. Z. Wong, H. Chen, S. Lin, and D. C. Chen, "Imputing missing values in sensor networks using sparse data representations," in *Proceedings of the 17th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, 2014, pp. 227–230.

[218] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in neural information processing systems*, 2001, pp. 556–562.

[219] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, p. 440, 1998.

[220] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *ICML*, 2017.

[221] Y.-J. Hong, I.-J. Kim, S. C. Ahn, and H.-G. Kim, "Activity recognition using wearable sensors for elder care," in *2008 Second International Conference on Future Generation Communication and Networking*, vol. 2. IEEE, 2008, pp. 302–305.

[222] P. Lukowicz, A. Timm-Giel, M. Lawo, and O. Herzog, "Wearit@ work: Toward real-world industrial wearable computing," *IEEE Pervasive Computing*, vol. 6, no. 4, pp. 8–13, 2007.

[223] A. Craik, Y. He, and J. L. Contreras-Vidal, "Deep learning for electroencephalogram (eeg) classification tasks: a review," *Journal of neural engineering*, vol. 16, no. 3, p. 031001, 2019.

[224] K. Hara, D. Saitoh, and H. Shouno, "Analysis of dropout learning regarded as ensemble learning," in *International Conference on Artificial Neural Networks*. Springer, 2016, pp. 72–79.

[225] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[226] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5188–5196.

[227] A. Dosovitskiy and T. Brox, "Generating images with perceptual similarity metrics based on deep networks," in *Advances in neural information processing systems*, 2016, pp. 658–666.

[228] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[229] T. Hossain, H. Goto, M. A. R. Ahad, and S. Inoue, "A study on sensor-based activity recognition having missing data," in *2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*. IEEE, 2018, pp. 556–561.

[230] J. Zhou and Z. Huang, "Recover missing sensor data with iterative imputing network," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[231] B. Fekade, T. Maksymyuk, M. Kyryk, and M. Jo, "Probabilistic recovery of incomplete sensed data in iot," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2282–2292, 2017.

[232] F. Imani, C. Cheng, R. Chen, and H. Yang, "Nested gaussian process modeling for high-dimensional data imputation in healthcare systems," in *IISE 2018 Conference & Expo, Orlando, FL, May*, 2018, pp. 19–22.

[233] X.-Y. Huang, W. Li, K. Chen, X.-H. Xiang, R. Pan, L. Li, and W.-X. Cai, "Multi-matrices factorization with application to missing sensor data imputation," *Sensors*, vol. 13, no. 11, pp. 15 172–15 186, 2013.

[234] V. F. Rey and P. Lukowicz, "Label propagation: An unsupervised similarity based method for integrating new sensors in activity recognition systems," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 3, pp. 1–24, 2017.