UNIVERSITY OF CALIFORNIA,
IRVINE

Context-Aware and Energy-Efficient Autonomous Systems

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Arnav Vaibhav Malawade

Dissertation Committee:
Professor Mohammad Abdullah Al Faruque, Chair
Associate Professor Marco Levorato
Assistant Professor Yasser Shoukry

2023

DEDICATION

To my parents, Vaibhav and Mathangi, for their unwavering love and support. To my sister, Krupa, for always being there for me. To my friends for their encouragement and inspiration. To Tessa, for her love, patience, and belief in me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Page

ACKNOWLEDGMENTS

I thank Professor Al Faruque, Professor Levorato, and Professor Shoukry for their feedback, mentorship, and support. I enjoyed working with them and appreciate their guidance throughout my Ph.D.

I collaborated with several colleagues during my graduate studies on various research projects. Specifically, I would like to thank Shih-Yuan Yu, Trier Mortlock, Mohanad Odema, and Yifan Zhang for their contributions to the projects presented in this dissertation. I learned a great deal working with them, and they helped shape my understanding of adjacent fields. I also want to thank all the undergraduate students I supervised who helped me complete these projects, including Nathan Costa, Preston Rogers, Ken Tran, Brandon Hsu, Sebastien Lajeunesse-Degroot, Harsimrat Kaeley, Anurag Karra, and Xiangbo Gao.

I also thank one of my mentors, Deepan Muthirayan, for consistently inspiring and challenging me via deep, thought-provoking brainstorming sessions. I also want to thank the senior Ph.D. students who helped guide my research when I first began on my path. Specifically, Sina Faezi, Sujit Chhetri, and Anthony Lopez were valuable sources of wisdom and helped me develop the skills needed to do productive and impactful research.

I would also like to thank the National Science Foundation for their funding assistance via awards CMMI-1739503, ECCS-1839429, and CCF-2140154, as well as Graduate Assistance in Areas of National Need (GAANN) for their funding assistance via award P200A180052. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding sources.

Acknowledgements to Publishers of Previous Work for Their Permission

Chapter 2 ©2022 IEEE. Reprinted, with permission, from Arnav Vaibhav Malawade, Shih-Yuan Yu, Brandon Hsu, Deepan Muthirayan, Pramod Khargonekar, Mohammad Abdullah Al Faruque, ”Spatiotemporal Scene-Graph Embedding for Autonomous Vehicle Collision Prediction”, IEEE Internet of Things Journal, January 2022.

VITA

Arnav Vaibhav Malawade

EDUCATION

Doctor of Philosophy in Electrical and Computer Engineering **2023**
University of California, Irvine *Irvine, California*

Master of Science in Electrical and Computer Engineering **2021**
University of California, Irvine *Irvine, California*

Bachelor of Science in Computer Science and Engineering **2018**
University of California, Irvine *Irvine, California*

RESEARCH EXPERIENCE

Doctorate Intern **2022–2022**
HRL Laboratories LLC *Malibu, California*

Graduate Research Assistant **2018–2023**
University of California, Irvine *Irvine, California*

Undergraduate Research Assistant **2018–2018**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant **2022–2022**
University of California, Irvine *Irvine, California*

Teaching Assistant **2018–2019**
University of California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

RS2G: Data-Driven Scene-Graph Extraction and Embedding for Robust Autonomous Perception and Scenario Understanding *(Under Review)* **2023**
Knowledge-Based Systems

CASTNet: A Context-Aware, Spatio-Temporal Dynamic Motion Prediction Approach for Autonomous Vehicles *(Under Review)* **2023**
ACM Transactions on Cyber-Physical Systems

roadscene2vec: A Tool for Extracting and Embedding Road Scene-Graphs **2022**
Knowledge-Based Systems

Spatio-Temporal Scene-Graph Embedding for Autonomous Vehicle Collision Prediction **2022**
IEEE Internet of Things Journal

Scene-Graph Augmented Data-Driven Risk Assessment of Autonomous Vehicle Decisions **2021**
IEEE Transactions on Intelligent Transportation Systems

SAGE: A Split-Architecture Methodology for Efficient End-to-End Autonomous Vehicle Control **2021**
ACM Transactions on Embedded Computing Systems

Neuroscience Inspired Algorithms for the Predictive Maintenance of Manufacturing Systems **2021**
IEEE Transactions on Industrial Informatics

Sabotage Attack Detection for Additive Manufacturing Systems **2020**
IEEE Access

Channel State Information-Based Cryptographic Key Generation for Intelligent Transportation Systems **2020**
IEEE Transactions on Intelligent Transportation Systems

Security of Emergent Automotive Systems: A Tutorial Introduction and Perspectives on Practice **2019**
IEEE Design & Test

REFEREED CONFERENCE PUBLICATIONS

CARMA: A Context-Aware Runtime Model Reconfiguration System for Energy-Efficient Sensor Fusion *(Under Review)* **2023**
ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)

EcoFusion: Energy-Aware Adaptive Sensor Fusion for Efficient Autonomous Vehicle Perception **2022**
ACM/IEEE Design Automation Conference (DAC)

HydraFusion: Context-Aware Selective Sensor Fusion for Robust and Efficient Autonomous Vehicle Perception **2022**
ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)

Multimodal Knowledge Graph for Deep Learning Papers and Code **2020**
Conference on Information and Knowledge Management (CIKM)

Oligo-Snoop: A Non-Invasive Side Channel Attack Against DNA Synthesis Machines **2019**
Network and Distributed System Security Symposium (NDSS)

SOFTWARE

roadscene2vec `https://github.com/AICPS/roadscene2vec`
An open-source tool for extracting and embedding road scene-graphs.

RS2G `https://github.com/AICPS/RS2G`
Open-source code for RS2G: a data-driven scene-graph extraction and embedding methodology.

HydraFusion `https://github.com/AICPS/hydrafusion`
Open-source PyTorch architecture for the HydraFusion model.

ABSTRACT OF THE DISSERTATION

Context-Aware and Energy-Efficient Autonomous Systems

By

Arnav Vaibhav Malawade

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2023

Professor Mohammad Abdullah Al Faruque, Chair

Autonomous systems (AS) are beginning to play a significant role in modern society. Examples of AS include robotic systems that perceive and react to changes in their environment, such as aerial drones, ground/aquatic robots, and consumer autonomous vehicles (AVs), among others. Breakthroughs in deep learning and increased consumer interest in ubiquitous autonomy have fueled recent advances in perception, modeling, and control algorithms. However, these advances come with rising energy costs. Modern AS require large deep-learning (DL) models to perceive the environment and safely detect and avoid objects. The computational demands of these models significantly increase the hardware requirements of the complete system, such that modern AV systems can require several kilowatts of power to enable autonomy, reducing range and utility. These impacts affect most AS since AVs, ground robots, and drones are typically edge devices that operate in energy-constrained environments.

The need for large, complex DL models is primarily driven by the fact that, in the real world, challenging and unpredictable scenarios can occur and must be modeled appropriately by the AS. However, existing approaches are exceedingly cautious, preferring to execute a large, inefficient model even in typical scenarios just in case a difficult edge case arises. This approach to autonomy hinders both the utility of existing AS and the broad-scale adoption of

AS. Instead, an AS should be able to understand the context of its surrounding environment and adapt its model to fit each situation.

This dissertation explores methods for developing context-aware AS to improve perception and prediction performance, reduce energy consumption, and enable adaptation to dynamic environments. Three different perspectives are studied: (i) scene-graph embedding approaches are proposed and evaluated for improved semantic understanding and state estimation; (ii) methods for implementing split ML models for dynamic, resource-efficient computation offloading from edge to cloud within real-time latency constraints are studied; and (iii) context-aware, dynamic ML perception models that jointly optimize performance and energy efficiency are studied from an algorithmic and system-wide optimization perspective. Overall, the results show that context-aware models achieve state-of-the-art performance across applications, and dynamic architectures conditioned on context can help resolve the energy-performance trade-off by enabling the joint optimization of these objectives.

Chapter 1

Introduction

1.1 Autonomous Systems

Autonomous systems (AS) are systems that can change behavior in response to unanticipated events [2]. Colloquially, this term refers to systems that operate without human control except for assigning a high-level objective (e.g., telling a robot to navigate to a location vs. having a human steer the robot). AS are expected to radically improve productivity, logistics, and safety by enabling aerial drones, ground/aquatic robots, and consumer autonomous vehicles (AVs) to operate without direct human control. Support for building AS is broad, with many looking forward to benefits such as automated drone delivery, undersea and space exploration, and improved search and rescue. AVs, in particular, are expected to improve road safety, passenger comfort, and mobility significantly [3, 4]. These applications require closely-coupled perception, state estimation, path planning, and control algorithms to safely maneuver the robot across complex and unpredictable real-world scenarios in real-time. Advances in deep learning, hardware design, and modeling over the past decade have made the dream of AS closer than ever to becoming a reality. However, the path to developing

AS has been anything but straightforward. A broad spectrum of AS approaches has been tried and tested with varying degrees of efficacy, ranging from rule-based expert systems to fully deep-learning-based methods. Most modern AS use one of two computing paradigms: (i) the modular architecture or (ii) the end-to-end architecture [3]. We study end-to-end architectures in Chapter 3 and study the perception module of modular architectures in Chapters 2, 4, 5.

Modular architectures are implemented as a pipeline of separate components for performing each sub-task of the AS (e.g., perception, localization, planning, control) [5, 3]. In contrast, end-to-end architectures generate actuator outputs (e.g., steering, brake, accelerator, motor control) directly from their sensory inputs [6]. One advantage of a modular design approach is that it divides the driving task into an easier-to-solve set of sub-tasks addressed in other fields, such as robotics, computer vision, and vehicle dynamics, from which prior knowledge can be leveraged. However, one disadvantage of such an approach is the complexity of implementing, executing, and validating the complete pipeline [3]. Worse, these approaches typically make assumptions at design time that may not scale to diverse real-world scenarios where edge cases are prevalent.

End-to-end approaches can achieve good performance with smaller network size and low implementation costs because they implicitly perform feature extraction from sensor inputs through the network's hidden layers [6]. This design also enables end-to-end models to share intermediate features between subtasks implicitly through the hidden layer features, which can improve overall modeling capability. However, the authors in [7] point out that the needed level of supervision is too weak for end-to-end models to learn critical control information (e.g., from image to steering angle), so they can fail to handle complicated driving maneuvers or be insufficiently robust to disturbances. These approaches also typically require vast datasets, increasing training time and associated costs, and result in a black-box model that makes decisions that are unexplainable and thus untrustworthy [8]. Therefore, modern

AS developers tend to prefer modular architectures.

Over the past decade, the supporting technologies (*e.g.*, sensors, hardware platforms, algorithms) have developed considerably, and companies such as Waymo [9], Tesla [10], and Baidu [11] have been developing and refining AS solutions using these approaches. However, AVs have yet to prove safe for consumers, supported by numerous reports of AV crashes [12, 13, 14]. Additionally, smaller AS, such as autonomous drones and ground robots, have only recently seen broader deployment. These facts raise a crucial question, what makes developing safe, reliable AS so challenging?

The answer is real-world driving scenarios are often unpredictable, complex, and difficult to replicate in simulations and challenging to account for at design time. Autonomous models are typically trained on large synthetic datasets because developers can easily simulate many operating conditions, environment types, and interaction scenarios to create an extensive database to train a driving model. Despite recent advances, a misalignment exists between synthesized datasets' scope and real-world driving complexity. This challenge is often called the *Sim2Real* gap [15]. *Sim2Real* is a term that describes the capability of a robotic system to effectively transfer knowledge gained from simulation environments to real-world applications [15]. Additional aspects that make real-world autonomy difficult to model include visual differences between sensor inputs, regional differences in road/environment layout, and interactions with unpredictable agents such as humans and animals. Negotiating intersections, crosswalks, and stop signs with human-driven vehicles and pedestrians proves challenging as an AV must understand the intent and behavior of the other agents in the environment. Thus, an AS should be able to (i) effectively generalize across scenarios and datasets and (ii) model interactions between agents to understand the environment better.

Notably, most current modular and end-to-end approaches utilize rigid model architectures that can limit adaptability to new driving scenarios. As a result, existing AS solutions are overly conservative, relying on exceedingly large and inefficient models to improve generalization and performance across scenarios. However, since most AS are designed to be mobile systems (e.g., AVs, drones, small ground robots), they have a fixed battery capacity and are thus energy-constrained edge devices. This constraint limits the size of AS algorithms to what can feasibly execute on the edge hardware platform for a reasonable operating time. Although these constraints are significant, energy efficiency is often overlooked in AS.

1.2 Energy Constraints in Autonomous Systems

Compared to a conventional robotic system, an AS requires multiple heterogeneous sensors (e.g., camera, radar, lidar) to collect data about the surroundings, large sensor processing and deep learning models to infer the current and future states of the environment, and large high-speed hardware computation platforms to execute these models within real-time latency constraints. The increasing complexity of modern AS comes with rising energy costs [5]. For example, the Nvidia Drive PX2, used in 2016-2018 Tesla models for their Autopilot system [10], can achieve 12 Tera Operations Per Second (TOPS) with a Thermal Design Power (TDP) of 250 Watts (W). Following the PX2 was the Nvidia AGX Pegasus, built for level 5 autonomy; it can achieve 320 TOPS with a TDP of 500W [16]. Moreover, the next-generation hardware platform using the Nvidia AGX Orin SoC is expected to be capable of 2000 TOPS with a TDP of 800W [17]. Although AV hardware platforms are becoming more efficient in TOPS/W, the baseline energy demands continue to increase as more advanced DL models and hardware platforms are developed. Additionally, current state-of-the-art radar and lidar sensors consume over 24W and 12W, respectively, and modern AVs use multiple of each sensor [18, 19]. The combined sensing, computation, and thermal loads can reduce the operational range of an AV by over 11.5% [5, 20]. Smaller AS (e.g., drones, ground robots) are more impacted due to their smaller battery capacity. For example, Boston Dynamics' Spot is a four-legged ground robot with a typical battery life of just 90 minutes [21], and current DJI drones have a flight time of only 30 minutes [22]. This issue is critical since most AS are energy-constrained edge devices [23, 24, 25, 26]. Moreover, reduced battery life necessitates more frequent recharging, which can negatively impact battery longevity long-term [27, 28, 29, 30, 31].

Researchers attempting to address this problem for AS have proposed several approaches for reducing energy consumption, including application-specific hardware design, cloud/fog server offloading, or model simplification/pruning [5, 32, 33, 29, 34, 35]. Although solutions like Application-Specific Integrated Circuits (ASICs) can reduce energy consumption through hardware optimization, they are prohibitively expensive to develop. Furthermore, with ASIC designs, all model specifications and contingencies must be accounted for at design time, meaning there is little to no support for adding new features, fixing algorithmic errors, or modifying model architectures. Costly development stages will need to be repeated for every revision to the model. The next logical choice is to attempt model simplification/pruning without changing hardware platforms; however, it is difficult to significantly reduce energy consumption by pruning without adversely affecting the AV's performance and safety. To address the limitations of the previous two approaches, some works propose offloading some or all AS tasks from the edge platform to the cloud for processing to reduce the energy consumption of the AS without changing the hardware or algorithms. Unfortunately, current offloading approaches have significant scalability and latency issues with the existing network infrastructure. Meanwhile, current clock and power gating methods are not flexible enough to adapt to different scenarios, so performance can suffer in difficult conditions. Overall, resolving the trade-off between energy efficiency and performance is still an open research problem. Addressing these issues necessitates a model capable of *adapting* to changing conditions to enable both energy efficiency and performance depending on the situation. Next, we discuss how *context* can help solve this problem.

1.3 Context-Awareness and Dynamic Neural Networks

Context is a set of facts or circumstances surrounding an event, situation, or object [36]. These facts are critically important to human cognition since humans naturally use context to improve their understanding of complicated scenarios and learn how to adapt to new situations. Intuitively, the performance of an AS depends on its ability to model context and understand scenarios since spatial awareness and scene understanding are critical to mission tasks such as motion planning and obstacle avoidance. However, traditional methods often only implicitly utilize context as part of the sensory input perceived by the model. If an AS can instead capture and model contextual information directly, it can improve its scene understanding and, thus, its performance across diverse scenarios and environments. Furthermore, context can enable AS to *adapt* to changing scenarios to improve robustness and energy efficiency compared to static approaches. The next question is, what kinds of contextual information exist?

An AS can leverage several different types of contextual information. Each type has utility in different kinds of problems. For example, [37] used three types of context for object detection: (i) likelihood of an object existing in a category of location, (ii) relative sizes of objects, and (iii) positional relations between objects. Contextual information can even take the form of inter-agent relationships [38, 39, 40] in settings with multiple dynamic agents (*e.g.*, driving through a busy intersection). [41] suggests that a human's ability to understand complex scenarios and identify potential risks relies on cognitive mechanisms for representing structure and reasoning about inter-object relationships. This form of context is used in Chapter 2. More broadly, weather, location type, and environmental features (e.g., street signs, traffic controls, pedestrian distance to the curb) can all be useful contextual information for an AS[42]. Features extracted from sensor data can also be used as contextual information, as in [43]. This approach, used in Chapters 4 and 5, is distinct from the implicit context identification of conventional models because the feature extractor is explicitly trained to distinguish between feature spaces and adapt the model. As a result, the features extracted represent a global state estimate inferred across all sensory inputs and can thus improve scene understanding. Extending to the vehicular networking (e.g., V2V, V2I, V2X) domain, the types of context can include the number and type of connected nodes, channel interference, and network throughput [44]. We use this form of context in Chapter 3. Although many types of contextual information exist, only a few works have proposed methods for creating context-aware machine learning models.

Context-aware machine learning approaches have proven effective for several applications. In these works, the context is typically modeled by passing the contextual information as an additional set of input features to the ML model. In [45], modeling human-to-human and human-to-space interactions as contextual features for an LSTM improves human trajectory prediction performance. Context has also proven effective in medical applications, where machine settings were used to determine if the sensor data is normal or abnormal, which is useful when a sensor can move out of place if the patient changes positions [46]. Context has also proven helpful for eliminating noise, fusing sensors, and improving the accuracy of small ML models across different human behaviors in mobile, wearable devices [47, 48, 49, 50]. Despite these successes in adjacent fields, context-aware approaches for implementing AS are under-studied. Contextual information such as driving environment and kinematics have also proven useful for energy optimization. [51] proposes dynamically adjusting an AV compute platform's scheduling parameters and processor speeds to meet dynamic deadlines that minimize energy usage. The dynamic deadlines are calculated using factors such as vehicle acceleration and driving context. Similarly, [52] proposes dynamically adjusting an AV lidar's power and operation state depending on environmental factors and vehicle speed to improve energy efficiency.

A key benefit of context modeling is enabling intelligent adaptation of deep neural network architectures. Compared to traditional, static models with fixed parameters and computation flow, dynamic networks can adapt and change structure to optimize different objectives at runtime [53]. Compared to static methods, this capability gives dynamic networks improved energy efficiency, representation power, adaptiveness, and generality. This dissertation explores how dynamic architectures conditioned on contextual information can enable energy efficiency and cross-scenario robustness via adaptation. Specifically, Chapters 4 and 5 explore *dynamic-width* sensor fusion networks, while Chapter 3 explores *dynamic routing* networks. Dynamic width refers to networks where the number of parallel processing submodels can increase or decrease. In contrast, dynamic routing refers to models that dynamically change the computation path with the same amount of processing. Other variants include *dynamic depth* networks, where the computation path can be exited early if the model is confident enough [54]. A complete taxonomy of dynamic networks is presented in [53]. Next, we discuss the research scope of this dissertation.

1.4 Research Scope

Fundamentally, current AS solutions are rigid, domain-specific, and energy-inefficient. Existing solutions only address one or two sub-problems at a time, while all three constraints must be considered to develop truly advanced and dependable AS. This dissertation addresses these challenges by exploring methods for implementing context-aware autonomous systems that can dynamically adapt to changing environments, save energy without compromising performance, and enable human-like understanding. Each chapter presents a new method for addressing these challenges in different AS applications:

- *Chapter 2* studies how graph representations of road scenes can improve scene understanding for collision prediction. In this case, the contextual information is embedded in the graph edges that model relationships between objects in a scene, enabling a more human-like understanding of how changing inter-object relationships relate to collision risk.

- *Chapter 3* studies how a split-architecture deep-learning model can enable energy-efficient end-to-end AV control via low latency computation offloading. The current networking conditions are used to decide when to make offloading decisions.

- *Chapter 4* studies how a dynamic width context-aware sensor fusion model can enable energy-efficient sensor fusion across diverse driving contexts. The proposed method first identifies the current driving context, then selects the best downstream model configuration to perceive in the current context. Here context can be external factors defined by domain knowledge (e.g., weather, lighting, road type). Additionally, context can be an abstract state estimate inferred from sensor data and encoded in the hidden representation of a deep-learning model.

- *Chapter 5* explores how the approach from Chapter 4 can be extended to a system-wide energy optimization. It explores how model reconfiguration on an FPGA, sensor clock gating, and intermittent context identification can enable greater energy savings.

- *Chapter 6* presents overall findings from these studies, elaborates on limitations, and discusses potential future research directions.

Though these problems affect all AS, we specifically study the benefits of these approaches for AVs due to the large scale of these systems, the challenge associated with interacting with human road users, and the direct benefits associated with developing AVs from safety, logistics, and mobility standpoints. However, it should be noted that the proposed methods can be applied to all comparable AS since the core tasks (*e.g.*, perception, planning, control) and enabling factors (*e.g.*, heterogeneous sensors, energy-constrained edge hardware, dynamic environments) are shared across systems.

Chapter 2

Scene-Graph Embedding for Collision Prediction

2.1 Introduction

Recent reports of AV crashes indicate that the development of safe and robust AVs remains a difficult challenge. For example, multiple fatal Tesla Autopilot crashes can primarily be attributed to perception system failures [12, 13]. Additionally, the infamous fatal collision between an Uber self-driving vehicle and a pedestrian can be attributed to perception and prediction failures by the AV [14]. These accidents (among others) have eroded public trust in AVs, and nearly 50% or more of the public have expressed their mistrust in AVs [4]. Current statistics indicate that perception and prediction errors were factors in over 40% of driver-related crashes between conventional vehicles [55]. However, a significant number of reported AV collisions are also the result of these errors [56, 57].

2.1.1 Research Challenges

Currently, most AV perception architectures rely entirely on either (i) deep learning techniques using Convolutional Neural Networks (CNNs) and Multi-Layer Perceptrons (MLPs) [58, 6, 59, 60]; or (ii) model-based methods, which use known road geometry information and vehicle trajectory models to estimate the state of the road scene [61, 62]. Although these approaches have been successful in typical use cases, they cannot obtain a high-level, human-like understanding of complex road scenarios. This limitation is due to their inability to explicitly capture inter-object relationships or the overall structure of the road scene. Understanding these relationships could be critical as it is suggested that a human's ability to understand complex scenarios and identify potential risks relies on cognitive mechanisms for representing structure and reasoning about inter-object relationships [41]. These models also require large datasets that are often costly or unsafe to generate. Synthetic datasets are typically used to augment the limited real-world data to train the models in such cases [63]. However, these trained models must then be able to transfer the knowledge gained from synthetic datasets to real-world driving scenarios. Furthermore, DL models contain millions of parameters and require IoT edge devices with significant computational power and memory to run efficiently. Likewise, hosting these models on the cloud is infeasible because it requires persistent low-latency internet connections. In summary, the key research challenges associated with autonomous vehicle perception are:

1. Capturing complex relationships between road users.
2. Accurately perceiving current and future risk factors to enable the AV to take corrective actions to avoid dangerous situations.
3. Generalizing to a wide range of traffic scenarios.
4. Developing algorithms that can run efficiently on automotive IoT edge devices.

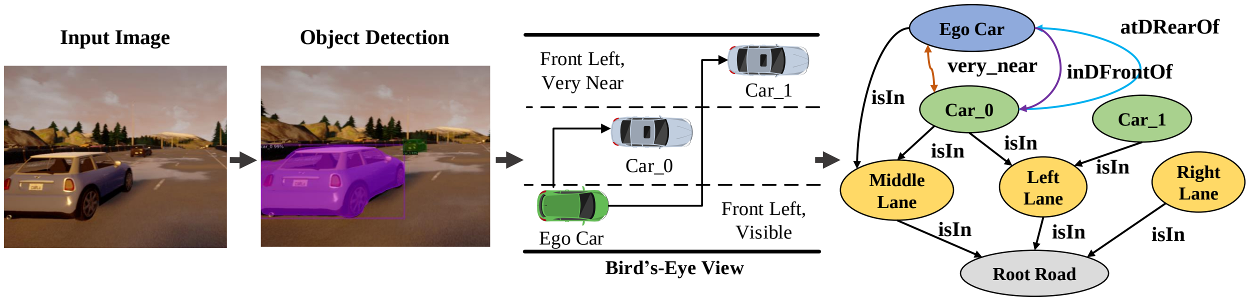


Figure 2.1: How camera data can be used to construct a road *scene-graph* representation.

2.1.2 Novel Contributions

To address these challenges, we propose **sg2vec**, which uses *scene-graphs* as intermediate representations (IR) of road scenes that explicitly model inter-object relationships to improve perception and scene understanding. Recently, several works have shown that graph-based methods that capture and model complex relationships between entities can improve performance at high-level tasks such as behavior classification [64, 65] and semantic segmentation [66]. A *scene-graph* representation encodes rich semantic information of an image or observed scene, essentially bringing an abstraction of objects and their complex relationships as illustrated in Figure 2.1. While each of these related works proposes a different form of *scene-graph* representation, all demonstrate significant performance improvements over conventional perception methods.

The *scene-graph* representation we propose represents traffic objects as nodes and the relationships between them as edges. The novelty of sg2vecs *scene-graph* representation lies in its graph construction technique that is specifically designed for higher-level scene understanding tasks such as collision prediction, as shown in Figure 2.2. We combine sg2vecs *scene-graph* representation with a spatio-temporal graph-embedding architecture to generate a sequence of *scene-graph* embeddings for the sequence of visual inputs perceived by an AV and predict the likelihood of a future collision. Overall, our novel contributions are as follows:

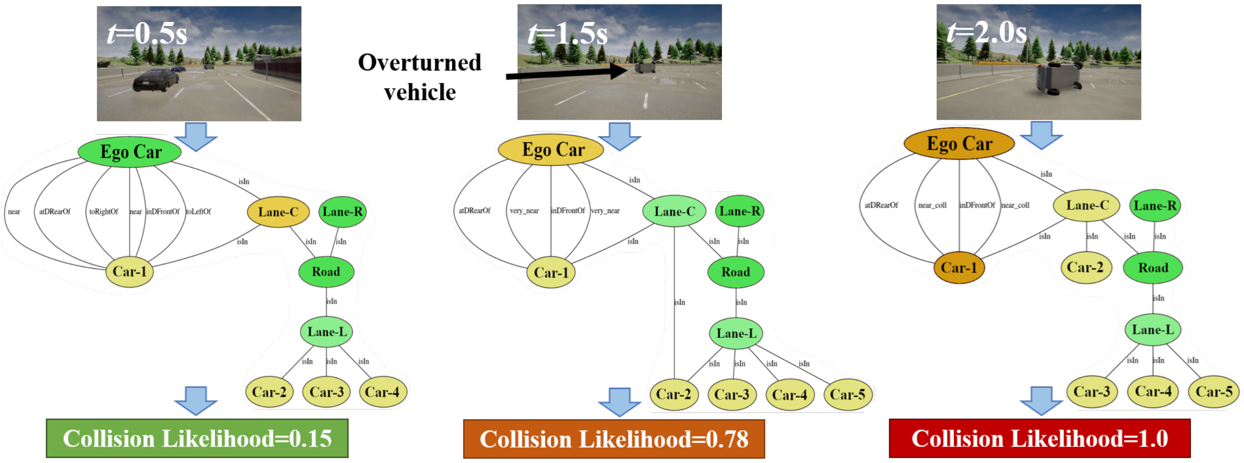


Figure 2.2: How sg2vec predicts collisions using *scene-graphs*. Each node's color indicates its attention score (importance to the collision likelihood) from orange (high) to green (low).

1. We propose **sg2vec**, an end-to-end graph construction and embedding approach for modeling *scene-graph* representations of road scenes.

2. We demonstrate how our approach significantly improves collision prediction performance over that of state-of-the-art methods on simulated lane-change datasets and a very challenging real-world collision dataset containing a wide range of driving actions, collision types, and weather/road conditions.

3. We empirically show that our approach transfers knowledge gained from simulated data to real-world driving data more effectively than the state of the art.

4. We demonstrate that our approach performs faster inference and requires less power than the state-of-the-art method on the industry-standard Nvidia DRIVE PX 2 autonomous driving hardware platform, used in 2016-2018 Tesla vehicles for their Autopilot system [10].

2.2 Related Work

2.2.1 Graph Learning

Graphs are composed of nodes and edges where the edges represent relations between the nodes. Graphs naturally exist in a wide range of real-world scenarios such as social graphs in social media networks, citation graphs in research areas, user-interest graph in electronic commerce area [67]. Graphs are also known to have complicated structures that contain rich underlying values [68]. Unlike images, audios, and texts which have a clear grid architecture, *graphs* are non-euclidean data that has irregular structures, making it hard to generalize some basic mathematical operations to *graphs*. As a result, how to utilize deep learning approaches for graph data has attracted considerable research attention in the past few years.

Most of the works in this field adopt the idea of *graph embedding* that aims to encode the structural information about the graph [69]. The main idea is to learn a mapping that embed nodes or the entire graphs as points in a low-dimensional vector space. The goal of these graph embedding methods is to optimize this mapping so that the geometric relationships in the embedding space reflect the structure of the original graph. Once this mapping is optimized, the learned embeddings of the graphs can be used as features inputs for downstream machine learning tasks.

The most popular approach for modeling the structure of graph data is the graph convolutional network (GCN), which implements *graph convolutions* [70]. Traditional CNNs leverage the regularity of image data structure, so the kernel is simply a rectangle that is swept across the image pixels. In contrast, the structure of each graph can be completely different, so the graph convolution kernel operates over the 1-hop edges to the center node. With multiple graph convolutions we can propagate across more hops. The convolution operation is commonly referred to as “message passing” [71]. To perform graph embedding, the resultant node embeddings are passed to a readout layer that condenses them into a single graph embedding. Then, tasks such as classification and regression can be done by simply passing this embedding to an MLP model. [72] presents several examples on how graph embedding can be applied across domains.

2.2.2 Graph-Based Scene Understanding

In contrast to existing methods that use CNN-based deep learning models for perception, we propose using a *scene-graph* IR that encodes the spatial and semantic relations between all the traffic participants in a frame. This form of representation is similar to a knowledge graph with the key distinction that *scene-graphs* explicitly encode knowledge about a visual scene. Several works have applied graph-based formulations for road scene understanding. In [73], the authors propose a 3D-aware egocentric Spatio-temporal interaction framework that uses both an *Ego-Thing* graph and an *Ego-Stuff* graph to encode how the ego vehicle interacts with both moving and stationary objects in a scene, respectively. In [64], the authors propose a pipeline using an MRGCN for classifying the driving behaviors of traffic participants. The MRGCN combines spatial and temporal information, including relational information between moving and landmark objects. In [74], the authors propose extracting road scene graphs in a manner that includes pose information for scene layout reconstruction. A similar approach was also proposed in [66]. Authors in [75] propose using a probabilistic graph approach for explainable traffic collision inference. Our prior work demonstrated that a *scene-graph* representation used with an MRGCN leads to state-of-the-art performance at assessing the subjective risk of driving maneuvers [38].

2.2.3 Collision Prediction and Avoidance

For the past several years, automotive manufacturers have begun equipping consumer vehicles with statistics-based collision avoidance systems based on calculated Single Behavior Threat Metrics (SBTMs) such as Time to Collision (TTC), Time to React (TTR), etc. [76, 77]. However, these methods lack robustness since they make significant assumptions about the behavior of vehicles on the road. A very limiting assumption they make is that vehicles do not diverge from their current trajectories [77]. SBTMs can also fail in specific scenarios. For example, TTC can fail when following a vehicle at the same velocity within a very short distance [77]. As a result, these methods are less capable of generalizing and can perform poorly in complex road scenarios. Moreover, to reduce false positives, these systems are designed to respond at the last possible moment [61]. Under such circumstances, the AV control system can fail to take timely corrective actions [78] if the system fails to predict a collision or estimates the TTC inaccurately.

Model-based probabilistic and deep learning approaches for collision prediction have also been proposed. For example, [79] proposes a model-based probabilistic technique that uses the roadway geometry, ego trajectory, and position/velocity of road objects to predict future object positions. However, this model is highly conservative and is likely to have a high false-positive rate. Similarly, [80] and [81] use model-based approaches but require significant domain knowledge about the driving scene, such as road geometry information as well as accurate vehicle position and velocity information. [82] proposes a deep learning collision prediction approach. Still, due to its use of pre-processed trajectory data captured from cameras overlooking a highway, it is not ego-centric and cannot be practically used for on-vehicle collision prediction. In a different approach, [83] proposes a Deep Predictive Model (DPM) that used a Bayesian Convolutional LSTM for collision risk assessment where image data, vehicle telemetry data, and driving inputs were all factors in the risk assessment decision. However, this approach was only evaluated on simulated street scenes containing two vehicles and no other dynamic objects. Thus, DPM's performance may suffer when evaluated on more complex road scenarios.

In contrast to these existing works, we propose sg2vec which captures structural and relational information of a road scene in a *scene-graph* representation and computes a spatio-temporal embedding to predict collisions. Additionally, we perform experiments that were not done in many prior works, such as evaluating each model's capability to transfer knowledge, efficiency on AV hardware, performance on a complex real-world crash dataset, and ability to predict collisions early. We primarily compare our methodology with the DPM as it is the state-of-the-art data-driven collision prediction framework for AVs that considers both spatial and temporal factors. Although the DPM uses multiple modalities for sensing, the results in [83] show that it achieves an accuracy (of 81.95%) that is just 0.24% less using just the image sensing modality. In this work, we compare our proposed sg2vec methodology and the DPM on image-only datasets, which is fair because the DPM's performance does not vary much with the inclusion of other modalities.

2.2.4 AV Power and Energy Optimization

Current autonomous driving systems consume a substantial amount of power (up to 500 Watts for the Nvidia DRIVE AGX Pegasus), demanding more robust cooling and power delivery mechanisms. Thus, many have tried to optimize AV tasks for efficiency without sacrificing performance. Existing approaches have proposed methods for jointly optimizing power consumption and latency for localization [84], perception [85], and control [86]. However, to the best of our knowledge, no work has explored this optimization for AV safety systems, such as collision prediction systems.

2.3 Scene-Graph Embedding Methodology

In sg2vec, we formulate the problem of collision prediction as a time-series classification problem where the goal is to predict if a collision will occur in the *near future.* Our goal is to accurately model the spatio-temporal function f, where

$$\mathbf{Y_n} = f(\{I_1, ..., I_{n-1}, I_n\}), \mathbf{Y_n} \in \{0, 1\}, \text{for } n > 2, \tag{2.1}$$

where $\mathbf{Y_n} = 1$ implies a collision in the near future and $\mathbf{Y_n} = 0$ otherwise. Here the variable I_n denotes the image captured by the on-board camera at time n. The interval between each frame varies with the camera sampling rate.

sg2vec consists of two parts (Figure 4.3) : (i) the *scene-graph* extraction, and (ii) collision prediction through spatio-temporal embedding, described in Section 2.3.1 and Section 2.3.2 respectively.

2.3.1 Scene-Graph Extraction

The first step of our methodology is the extraction of *scene-graphs* for the images of a driving scene. The extraction pipeline forms the *scene-graph* for an image as in [87, 88] by first detecting the objects in the image and then identifying their relations based on their attributes. The difference from prior works lies in the construction of a *scene-graph* that is designed for higher-level AV decisions. We propose extracting a *minimal* set of relations such as directional relations and proximity relations. From our design space exploration we found that adding many relation edges to the *scene-graph* adds noise and impacts convergence while using too few relation types reduces our model's expressivity. The best approach we found across applications involves constructing mostly ego-centric relations for a moderate range of relation types. Figure 2.1 shows an example of the graph extraction process.

We denote the extracted *scene-graph* for the frame I_n by $G_n = \{O_n, A_n\}$. Each *scene-graph* G_n is a directed, heterogeneous multi-graph, where O_n denotes the nodes and A_n is the adjacency matrix of the graph G_n. As shown in Fig. 2.1, nodes represent the identified objects such as lanes, roads, traffic signs, vehicles, pedestrians, etc., in a traffic scene. The adjacency matrix A_n indicates the pair-wise relations between each object in O_n. The extraction pipeline first identifies the objects O_n by using Mask R-CNN [89]. Then, it generates an inverse perspective mapping (also known as a "birds-eye view" projection) of the image to estimate the locations of objects relative to the ego car, which are used to construct the pair-wise relations between objects in A_n. For each camera angle, we calibrate the birds-eye view projection settings using known fixed distances, such as the lane length and width, as defined by the highway code. This enables us to estimate longitudinal and lateral distances accurately in the projection. For datasets captured by a single vehicle, this step only needs to be performed once. However, for datasets with a wide range of camera angles such as the *620-dash* dataset introduced later in the chapter, this process needs to be performed once per vehicle. With a human operator, we found that this calibration step takes approximately 1 minute per camera angle on average.

The extraction pipeline identifies three kinds of pair-wise relations: *proximity* relations (e.g. *visible*, *near*, *very_near*, etc.), *directional* (e.g. *Front_Left*, *Rear_Right*, etc.) relations, and *belonging* (e.g. car_1 *isIn* left_lane) relations. Two objects are assigned the *proximity* relation, $r \in \{$*Near_Collision* (4 ft.), *Super_Near* (7 ft.), *Very_Near* (10 ft.), *Near* (16 ft.), *Visible* (25 ft.)$\}$ provided the objects are physically separated by a distance that is within that relation's threshold. The *directional relation*, $r \in \{$*Front_Left*, *Left_Front*, *Left_Rear*, *Rear_Left*, *Rear_Right*, *Right_Rear*, *Right_Front*, *Front_Right*$\}$, is assigned to a pair of objects, in this case between the ego-car and another car in the view, based on their relative orientation and only if they are within the *Near* threshold distance from one another. Additionally, the *isIn* relation identifies which vehicles are on which lanes (see Fig. 2.1). We use each vehicle's horizontal displacement relative to the ego vehicle to assign vehicles to either the *Left Lane*, *Middle Lane*, or *Right Lane* using the known lane width. Our abstraction only considers three-lane areas, and, as such, we map vehicles in all left lanes and all right lanes to the same *Left Lane* node *Right Lane* node respectively. If a vehicle overlaps two lanes (i.e., during a lane change), it is mapped to both lanes.

2.3.2 Collision Prediction

As shown in Figure 4.3, in our collision prediction methodology, each image I_n is first converted into a *scene-graph* $G_n = \{O_n, A_n\}$ with the pipeline mentioned in Section 2.3.1. Each node $v \in O_n$ is initialized by a one-hot vector (*embedding*), denoted by $\mathbf{h}_v^{(0)}$. Then, the MRGCN [90] layers are used to update these embeddings via the edges in A_n. Specifically, the l-th MRGCN layer computes the node embedding for each node v, denoted as $\mathbf{h}_v^{(l)}$, as follows:

$$\mathbf{h}_v^{(l)} = \mathbf{\Phi}_0 \cdot \mathbf{h}_v^{(l-1)} + \sum_{r \in \mathbf{A_n}} \sum_{u \in \mathbf{N}_r(v)} \frac{1}{|\mathbf{N}_r(v)|} \mathbf{\Phi}_r \cdot \mathbf{h}_u^{(l-1)}, \tag{2.2}$$

where $N_r(v)$ denotes the set of neighbors of node v with respect to the relation $r \in A_n$, $\mathbf{\Phi}_r$ is a trainable relation-specific transformation for relation r, and $\mathbf{\Phi}_0$ is the self-connection for each node v that accounts for the influence of $\mathbf{h}_v^{(l-1)}$ on $\mathbf{h}_v^{(l)}$ [90]. After the input is passed through multiple MRGCN layers, the set of node embeddings output by each layer is collected and concatenated along the feature dimension to produce the final embedding for each node v, denoted by $\mathbf{H}_v^L = \mathbf{CONCAT}(\{\mathbf{h}_v^{(l)}\}|l = 0, 1, ..., L)$, where L is the index of the last layer. Thus, if the model uses two MRGCN layers with output size 64, the final embedding for each node will contain 128 features.

The final embeddings for *scene-graph* G_n, denoted by $\mathbf{X}_n^{prop}$, are then passed through a graph pooling layer to filter out irrelevant nodes from the graph, creating the pooled set of node embeddings $\mathbf{X}_n^{pool}$ and their edges $\mathbf{A}_n^{pool}$. The pooling layer is implemented as follows:

$$\alpha = \mathbf{SCORE}(\mathbf{X}_n^{prop}, \mathbf{A}_n^{prop}) \tag{2.3}$$

$$\mathbf{P} = \text{top}_k(\alpha) \tag{2.4}$$

where **SCORE** can either be implemented as a top-k pooling (*Top-K*) [91] or self-attention graph pooling function (*SAGPool*) [92], α contains the score of each node in G_n, and $\mathbf{P}$ is the set of k highest scoring nodes in G_n. After pooling, the node embeddings and adjacency matrix are denoted as $\mathbf{X}_t^{pool}$ and $\mathbf{A}_\mathbf{t}^{\mathbf{pool}}$ computed as follows:

$$\mathbf{X}_t^{pool} = (\mathbf{X}_t^{prop} \odot \tanh(\alpha))_\mathbf{P}, \tag{2.5}$$

$$\mathbf{A}_\mathbf{t}^{\mathbf{pool}} = \mathbf{A}_\mathbf{t}^{\mathbf{prop}}{}_{(\mathbf{P},\mathbf{P})} \tag{2.6}$$

where $\odot$ represents element-wise multiplication, $()_\mathbf{P}$ refers to the operation that selects only the subset of nodes defined by P and $()_{(\mathbf{P},\mathbf{P})}$ refers to the formation of the adjacency matrix between the nodes in this subset.

Then, for each *scene-graph* G_n, the corresponding $\mathbf{X}_n^{pool}$ is passed through the graph **READOUT** operation that condenses the node embeddings to a single graph embedding $\mathbf{h}_{G_n}$ as follows:

$$\mathbf{h}_{G_n} = \mathbf{READOUT}(X_t^{pool}) \tag{2.7}$$

where **READOUT** can be an operation such as averaging (*mean-readout*), summation (*add-readout*), or retrieving the maximum (*max-readout*) in each feature dimension for the set of pooled node embeddings X_t^{pool}.

Then, this spatial embedding $\mathbf{h}_{G_n}$ is passed to the temporal model (LSTM) to generate a spatio-temporal embedding z_n as follows:

$$z_n, s_n = \mathbf{LSTM}(h_{G_n}, s_{n-1}) \tag{2.8}$$

Where s_{n-1} represents the hidden state of the LSTM after the previous time step. For each timestamp n, the LSTM produces an output embedding z_n and updates its hidden state s_n. Since the hidden state is carried over to the next time step $n+1$ and used to compute z_{n+1}, it enables the LSTM to model how the spatial embeddings h_{G_n} change over time.

Lastly, each spatio-temporal embedding z_n is then passed through a Multi-Layer Perceptron (MLP) that outputs each class's confidence value. The two outputs of the MLP are compared, and $\hat{Y}_n$ is set to the index of the class with the greater confidence value (0 for no-collision or 1 for collision). During training, we calculate the cross-entropy loss between each set of non-binarized outputs $\hat{Y}_n$ and the corresponding labels for backpropagation.

2.4 Experimental Results

This section provides extensive experimental results to demonstrate sg2vec's performance, efficiency, and transferability compared to the state-of-the-art collision prediction model, DPM [83]. For sg2vec, we used 2 MRGCN layers, each of size 64, one *SAGPooling* layer with a pooling ratio of 0.25, one *add-readout* layer, one LSTM layer with hidden size 20, one MLP layer with an output of size 2, and a LogSoftmax to generate the final confidence value for each class. For the DPM, we followed the architecture used in [83], which uses one 64x64x5 Convolutional LSTM (ConvLSTM) layer, one 32x32x5 ConvLSTM layer, one 16x16x5 ConvLSTM layer, one MLP layer with output size 64, one MLP layer with output size 2, and a Softmax to generate the final confidence value. For both models, we used a dropout of 0.1 and ReLU activation. The learning rates were 0.00005 for sg2vec and 0.0001 for DPM. We ran the experiments shown in Sections 2.4.2 and 2.4.3 on a Windows PC with an AMD Ryzen Threadripper 1950X processor, 16 GB RAM, and an Nvidia GeForce RTX 2080 Super GPU.

2.4.1 Dataset Preparation

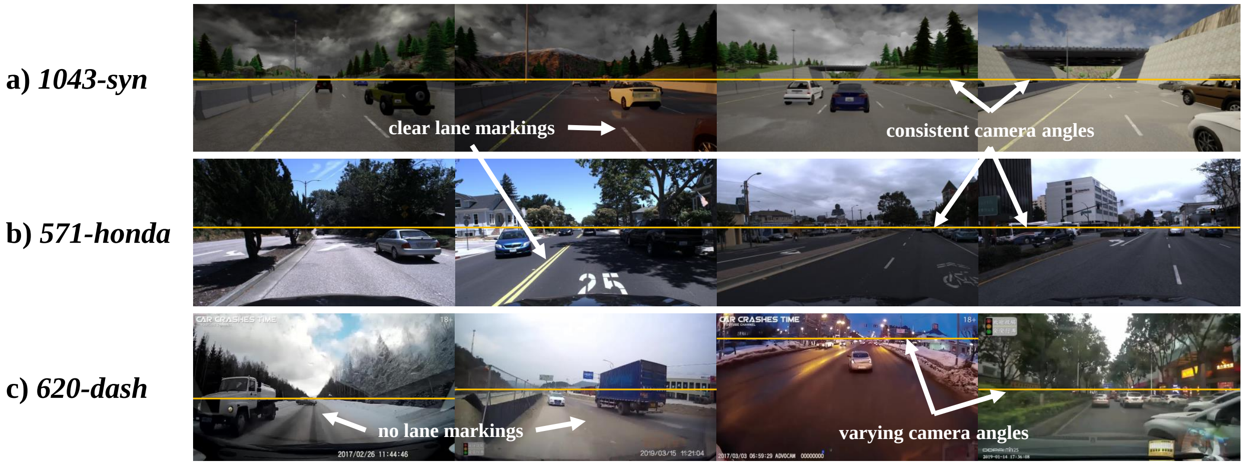


Figure 2.3: Examples of driving scenes from our a) synthetic datasets, b) typical real-world dataset, and c) complex real-world dataset. In a), all driving scenes occur on highways with the same camera position and clearly defined road markings; lighting and weather are dynamically simulated in CARLA. In b) driving scenes occur on multiple types of clearly marked roads but lighting, camera angle, and weather are consistent across scenes. c) contains a much broader range of camera angles as well as more diverse weather and lighting conditions, including rain, snow, and night-time driving; it also contains a large number of clips on unpaved or unmarked roadways, as shown.

We prepared three types of datasets for our experiments: (i) synthesized datasets, (ii) a typical real-world driving dataset, and (iii) a complex real-world driving dataset. Examples from each dataset are shown in Figure 2.3. Our synthetic datasets focus on the highway lane change scenario as it is a common AV task. To evaluate the transferability of each model from synthetic datasets to real-world driving, we prepared a typical real-world dataset containing lane-change driving clips. Finally, we prepared the complex real-world driving dataset to evaluate each model's performance on a challenging dataset containing a broad spectrum of collision types, road conditions, and vehicle maneuvers. All datasets were collected at a

1280x720 resolution, and each clip spans 1-5 seconds.

Synthetic Datasets

To synthesize the datasets, we developed a tool using CARLA [63], an open-source driving simulator, and CARLA Scenario Runner[1] to generate lane change video clips with/without collisions. We generated a wide range of simulated lane changes with different numbers of cars, pedestrians, weather and lighting conditions, etc. We also customized each vehicle's driving behavior, such as their intended speed, probability of ignoring traffic lights, or the chance of avoiding collisions with other vehicles. We generated two synthetic datasets: a *271-syn* dataset and a *1043-syn* dataset, containing 271 and 1,043 video clips, respectively. These datasets have no-collision:collision label distributions of 6.12:1 and 7.91:1, respectively. In addition, we sub-sampled the *1043-syn* dataset to create *306-syn*: a balanced dataset that has a 1:1 distribution. Our synthetic *scene-graph* datasets[2] and our source code[3] are open-source and available online. sg2vec is also now a part of the roadscene2vec library [93].

Typical Real-World Driving Dataset

This dataset, denoted as *571-honda*, is a subset of the Honda Driving Dataset (HDD) [94] containing 571 lane-change video clips from real-world driving with a distribution of 7.21:1. The HDD was recorded on the same vehicle during mostly safe driving in the California Bay Area.

[1] `https://github.com/carla-simulator/scenario_runner`

[2] `https://dx.doi.org/10.21227/c0z9-1p30`

[3] `https://github.com/AICPS/sg-collision-prediction`

Complex Real-World Driving Dataset

Our complex real-world driving dataset, denoted as *620-dash*, contains very challenging real-world collision scenarios drawn from the Detection of Traffic Anomaly dataset [95]. This dataset contains a wide range of drivers, car models, driving maneuvers, weather/road conditions, and collision types, as recorded by on-board dashboard cameras. Since the original dataset contains only collision clips, we prepared *620-dash* by splitting each clip in the original dataset into two parts: (i) the beginning of the clip until 1 second before the collision, and (ii) from 1 second before the collision until the end of the collision. We then labeled part (i) as 'no-collision' and part (ii) as 'collision.' The *620-dash* dataset contains 315 collision video clips and 342 non-collision driving clips.

Labeling and Pre-Processing

We labeled the synthetic datasets and the *571-honda* dataset using human annotators. The final label assigned to a clip is the average of the labels assigned by the human annotators rounded to 0 (no collision) and 1 (collision/near collision). Each frame in a video clip is given a label identical to the entire clip's label to train the model to identify the preconditions of a future collision.

For sg2vec, all the datasets were pre-processed using the *scene-graph* extraction pipeline mentioned in Section 2.3.1 to construct the *scene-graphs* for each video clip. For a given sequence, sg2vec can leverage the full history of prior frames for each new prediction. For the DPM, the datasets were pre-processed to match the input format used in its original implementation [83]. Thus, the DPM uses 64x64 grayscale versions of the clips in the datasets turned into sets of sub-sequences J_n for a clip of length l defined as follows.

$$J_n = \{I_n, I_{n+1}, I_{n+2}, I_{n+3}, I_{n+4}\}, \text{for } n \in [1, l-4] \tag{2.9}$$

Since DPM only uses five prior frames to make each prediction, we also present results for sg2vec using the same length of history, denoted as sg2vec (5-frames) in the results.

2.4.2 Collision Prediction Performance

We evaluated sg2vec and the DPM using classification accuracy, area under the ROC curve (AUC) [96], and Matthews Correlation Coefficient (MCC) [97]. MCC is considered a balanced measure of performance for binary classification even on datasets with significant class imbalances. The MCC score outputs a value between -1.0 and 1.0, where 1.0 corresponds to a perfect classifier, 0.0 to a random classifier, and -1.0 to an always incorrect classifier. Although class re-weighting helps compensate for the dataset imbalance during training, classification accuracy is typically less reliable for imbalanced datasets, so the primary metric we use to compare the models is MCC. We used stratified 5-fold cross-validation to produce the final results shown in Table 2.1 and Figure 2.4.

Synthetic Datasets

The performance of sg2vec and the DPM on our synthetic datasets is shown in Table 2.1. We find that our sg2vec achieves higher accuracy, AUC, and MCC on every dataset, even when only using five prior frames as input. In addition to predicting collisions more accurately, sg2vec also infers **5.5x** faster than the DPM on average. We attribute this to the differences in model complexity between our sg2vec architecture and the much larger DPM model. Interestingly, sg2vec (5-frames) achieves slightly better accuracy and AUC than sg2vec on

Dataset	Model	Accuracy	AUC	MCC
271-syn	sg2vec (5-frames)	**0.8979**	**0.9541**	**0.5362**
271-syn	sg2vec	0.8812	0.9457	0.5145
271-syn	DPM	0.8733	0.8939	0.2160
306-syn	sg2vec (5-frames)	0.7946	0.8653	0.5790
306-syn	sg2vec	**0.8372**	**0.9091**	**0.6812**
306-syn	DPM	0.6846	0.6881	0.3677
1043-syn	sg2vec (5-frames)	**0.9142**	**0.9623**	0.5323
1043-syn	sg2vec	0.9095	0.9477	**0.5385**
1043-syn	DPM	0.8834	0.9175	0.2912
620-dash	sg2vec (5-frames)	0.6534	0.7113	0.3053
620-dash	sg2vec	**0.7007**	**0.7857**	**0.4017**
620-dash	DPM	0.4890	0.4717	-0.0366

Table 2.1: Classification accuracy, AUC, and MCC for sg2vec (Ours) and DPM.

the imbalanced datasets and slightly lower overall performance on the balanced datasets. This is likely because the large number of safe lane changes in the imbalanced datasets adds noise during training and makes the full-history version of the model perform slightly worse. However, the full model can learn long-tail patterns for collision scenarios and performs better on the balanced datasets.

The DPM achieves relatively high accuracy and AUC on the imbalanced *271-syn* and *1043-syn* datasets, but suffers significantly on the balanced *306-syn* dataset. This drop indicates that the DPM could not identify the minority class (collision) well and tended to over-predict the majority class (no-collision). In terms of MCC, the DPM scores higher on the *306-syn* dataset than what it scores on the other datasets. This result is because the *306-syn* dataset has a balanced class distribution compared to the other datasets, which could enable the DPM to improve its prediction accuracy on the collision class.

In contrast, the sg2vec methodology performs well on both balanced and imbalanced synthetic datasets with an average MCC of **0.5860**, an average accuracy of **87.97%**, and an average AUC of **0.9369**. Since MCC is scaled from -1.0 to 1.0, sg2vec achieves a **14.72%** higher average MCC score than the DPM model.

The results from our sg2vec ablation study are shown in Table 2.3 and support our hypothesis that both spatial modeling with MRGCN and temporal modeling with LSTM are core to sg2vec's collision prediction performance. However, the MRGCN appears to be slightly more critical to performance than the LSTM. Interestingly the choice of pooling layer (no pooling, Top-K pooling, or SAG Pooling) does not seem to significantly affect performance at this task as long as LSTM is used; when no LSTM is used SAG Pooling presents a clear performance improvement.

Complex Real-World Dataset

The performance of both the models significantly drops on the highly complex real-world *620-dash* dataset due to the variations in the driving scenes and collision scenarios. This drop is to be expected as this dataset contains a wide range of driving actions, road environments, and collision scenarios, increasing the difficulty of the problem significantly. We took several steps to try and address this performance drop. First, we improved the birds-eye view (BEV) calibration on this dataset in comparison to the other datasets. Since the varying camera angles and road conditions in this dataset impact our ability to properly calibrate sg2vec's BEV projection in a single step, we created custom BEV calibrations for each clip in the dataset, which improved performance somewhat. However, as shown in Figure 4c, a significant part of the dataset consists of driving clips on roads without any discernible lane markings, such as snowy, unpaved, or unmarked roadways. These factors make it challenging to correlate known fixed distances (i.e., the width and length of lane markings) with the projections of these clips. To further improve performance on this particular dataset, we performed extensive architecture and hyperparameter tuning. We found that, with one MRGCN layer of size 64, one LSTM layer with hidden size 100, no SAGPooling layer, and a high learning rate and batch size, we achieved significantly better performance than the model architecture discussed in the beginning of Section 2.4 (2 MRGCN layers of size 64, one LSTM layer with hidden size 20, and a SAGPooling layer with a keep ratio of 0.5). We believe this indicates that the temporal features of each clip in this dataset are more closely related to collision likelihood than the spatial features in each clip. As a result, the additional spatial modeling components were likely causing overfitting and skewing the spatial embedding output. The spatial embeddings remained more general with a simpler spatial model (1 MRGCN and no SAGPooling). This change, combined with using a larger LSTM layer, enabled the model to capture more temporal features when modeling each clip and better generalize to the testing set. Model performance on this dataset and similar datasets could likely be improved by acquiring more consistent data via higher-resolution cameras with fixed camera angles and more accurate BEV projection approaches. However, as collisions are rare events, there are little to no datasets containing real-world collisions that meet these requirements. Despite these limitations, sg2vec outperforms the DPM model by a significant margin, achieving **21.17%** higher accuracy, **31.40%** higher AUC, and a **21.92%** higher MCC score. Since DPM achieves a negative MCC score, its performance on this dataset is worse than that of a random classifier (MCC of 0.0). Consistent with the synthetic dataset results, *sg2vec* using all frames performs better on the balanced *620-dash* dataset than sg2vec (5-frames). Overall, these results show that, on very challenging and complex real-world driving scenarios, sg2vec can perform much better than the current state-of-the-art.

Time of Prediction

Since collision prediction is a time-sensitive problem, we evaluated our methodology and the DPM on their average time-of-prediction (ATP) for video clips containing collisions. To calculate the ATP, we recorded the first frame index in each collision clip when the model correctly predicts that a collision would occur. We then averaged these indices and compared them with the average collision video clip length. Essentially, ATP gives an estimate of how early each model can predict a future collision. These results are shown in Table 2.2. On the *1043-syn* dataset, sg2vec achieves 0.1725 for the ratio of the ATP and the average sequence length while the DPM achieves a ratio of 0.2382, indicating that sg2vec predicts future collisions **39.07%** earlier than the DPM on average. In the context of real-world collision prediction, the average sequence in the *1043-syn* dataset represents 1.867 seconds of data. Thus, our methodology predicted collisions **122.7** milliseconds earlier than DPM on average. This extra time can be critical for ensuring that the AV avoids an impending collision.

Dataset	Model	ATP	Avg. Seq. Len.	Ratio
271-syn	**sg2vec (Ours)**	10.004	33.920	**0.2949**
271-syn	DPM	17.399	32.899	0.5289
1043-syn	**sg2vec (Ours)**	6.442	37.343	**0.1725**
1043-syn	DPM	9.018	37.856	0.2382

Table 2.2: Average time of prediction (ATP) for collisions.

Experiment	Spatial Model	Graph Pooling	Temporal Model	Acc.	MCC
Ablation Study	MLP	none	none	0.7605	0.2612
	MLP	none	LSTM	0.7660	0.2874
	MRGCN	none	none	0.8605	0.4792
	MRGCN	none	LSTM	**0.8931**	**0.5561**
Graph Attn. and Pooling	MRGCN	Top-K	none	0.8288	0.3458
	MRGCN	SAGPool	none	0.8738	0.5032
	MRGCN	Top-K	LSTM	**0.9014**	**0.5565**
	MRGCN	SAGPool	LSTM	**0.9076**	**0.5407**

Table 2.3: sg2vec ablation study on the *1043-syn* dataset.

2.4.3 Transferability From Synthetic to Real-World Datasets

The collision prediction models trained on simulated datasets must be transferable to real-world driving as it can differ significantly from simulations. To evaluate each model's ability to transfer knowledge, we trained each model on a synthetic dataset before testing it on the *571-honda* dataset. No additional domain adaptation was performed. Figure 2.4 compares the accuracy and MCC for both the models on each training dataset and the *571-honda* dataset after transferring the trained model.

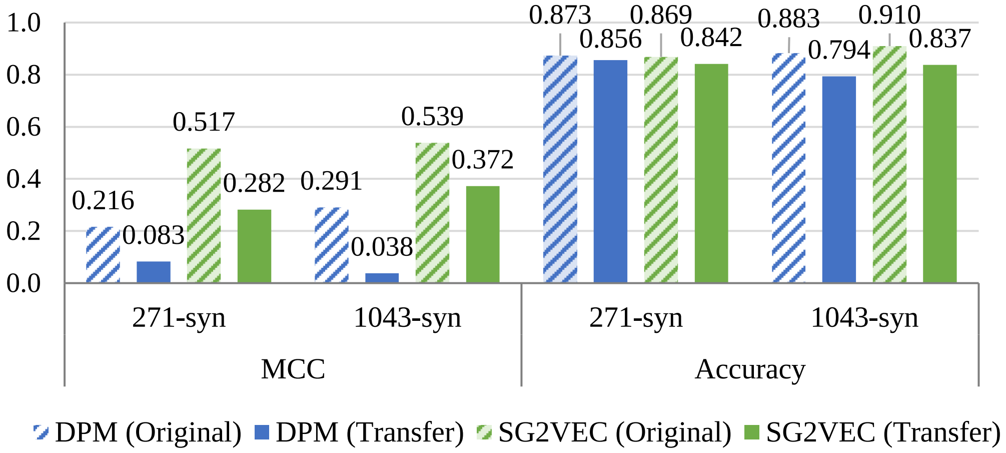


Figure 2.4: Performance after transferring the models trained on synthetic *271-syn* and *1043-syn* datasets to the real-world *571-honda* dataset.

We observe that the sg2vec model achieves a significantly higher MCC score than the DPM model after the transfer, suggesting that our methodology can better transfer knowledge from a synthetic to a real-world dataset compared to the state-of-the-art DPM model. The drop in MCC values observed for both the models when transferred to the *571-honda* dataset can be attributed to the characteristic differences between the simulated and real-world datasets; the *571-honda* dataset contains a more heterogeneous set of road environments, lighting conditions, driving styles, etc., so a drop in performance after the transfer is expected. We also note that the MCC score for the sg2vec model trained on *271-syn* dataset drops more than the model trained on the *1043-syn* dataset after the transfer, likely due to the smaller training dataset size. Regarding accuracy, the sg2vec model trained on *1043-syn* achieves 4.37% higher accuracy and the model trained *271-syn* dataset achieves 1.47% lower accuracy than the DPM model trained on the same datasets. The DPM's similar accuracy after transfer likely results from the class imbalance in the *571-honda* dataset. Overall, we hypothesize that sg2vec's use of an intermediate representation (i.e., *scene-graphs*) inherently improves its ability to generalize and thus results in an improved ability to transfer knowledge compared to CNN-based deep learning approaches.

2.4.4 Evaluation on Industry-Standard AV Hardware

To demonstrate that the sg2vec is implementable on industry-standard AV hardware, we measured its inference time (milliseconds), model size (kilobytes), power consumption (watts), and energy consumption per frame (milli-joules) on the industry-standard Nvidia DRIVE PX 2 platform, which was used by Tesla for their Autopilot system from 2016 to 2018 [10]. Our hardware setup is shown in Figure 2.5. For the inference time, we evaluated the average inference time (AIT) in milliseconds taken by each algorithm to process each frame. We recorded power usage metrics using a power meter connected to the power supply of the PX 2. To ensure that the reported numbers only reflected each model's power consumption and not that of background processes, we subtracted the hardware's idle power consumption from the averages recorded during each test. For a fair comparison, we captured the metrics for the core algorithm (i.e., the sg2vec and DPM model), excluding the contribution from data loading and pre-processing. Both models were run with a batch size of 1 to emulate the real-world data stream where images are processed as they are received. For comparison, we also show the AIT on a PC for the two models.

Our results are shown in Table 2.4. sg2vec performs inference **9.3x** faster than the DPM on the PX 2 with an **88.0%** smaller model and **32.4%** less power, making it undoubtedly more practical for real-world deployment. Our model also uses **92.8%** less energy to process each frame, which can be beneficial for electric vehicles with limited battery capacity. With an AIT of 0.4828 ms, sg2vec can theoretically process up to 2,071 frames/second (fps). In contrast, with an AIT of 4.535 ms, the DPM can only process up to 220 fps. In the context of real-world collision prediction, this means that sg2vec could easily support multiple 60 fps camera inputs from the AV while DPM would struggle to support more than three.

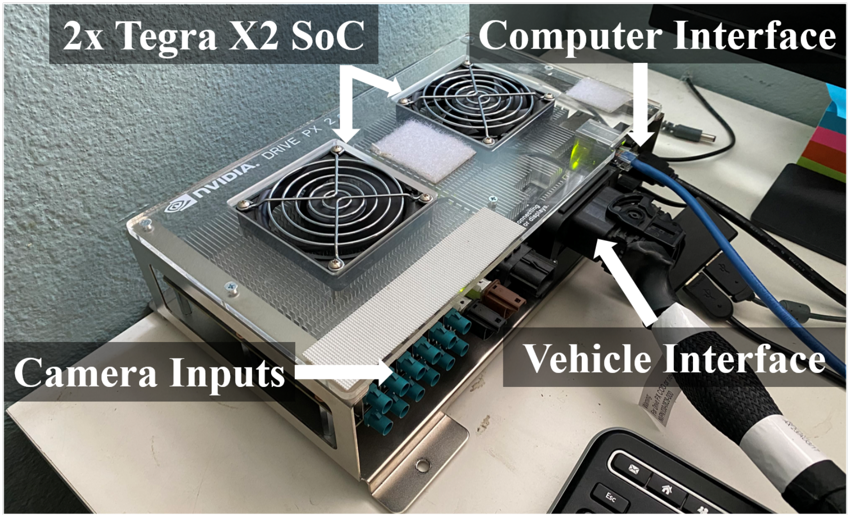


Figure 2.5: Our experimental setup for evaluating sg2vec and DPM on the industry-standard Nvidia DRIVE PX 2 hardware.

Model	PC AIT (ms)	PX2 AIT (ms)	Size (KB)	Power (W)	Energy/frame (mJ)
sg2vec	**0.2549**	**0.4828**	**331**	**2.99**	**1.44**
DPM	1.393	4.535	2,764	4.42	20.0

Table 2.4: Performance evaluation of inference on *271-syn* on the Nvidia DRIVE PX 2.

2.5 Summary

Our experiments demonstrate that our *scene-graph* extraction and embedding methodology for collision prediction, sg2vec, outperforms the state-of-the-art method, DPM, in terms of average MCC (0.5055 vs. 0.2096), average inference time (0.255 ms vs. 1.39 ms), and average time of prediction (39.07% sooner than DPM). Additionally, they show that sg2vec can transfer knowledge from synthetic datasets to real-world driving datasets more effectively than the DPM, achieving an average transfer MCC of 0.327 vs. 0.060. Finally, we find that our methodology performs faster inference than the DPM (0.4828 ms vs. 4.535 ms) with a smaller model size (331 KB vs. 2,764 KB) and reduced power consumption (2.99 W vs. 4.42 W) on the industry-standard Nvidia DRIVE PX 2 autonomous driving platform. In the context of real-world collision prediction, these results indicate that sg2vec is a more practical choice for AV safety and could significantly improve consumer trust in AVs. Few works have explored graph-based solutions for other complex AV challenges such as localization, path planning, and control. These are open research problems that we reserve for future work.

This chapter focused on improving scene understanding by modeling the context as inter-object relations in a scene-graph. Thus, transfer learning ability and robust performance were prioritized over energy optimization. The next chapter focuses much more closely on the energy problem and explores how dynamic networks can help traditional DL architectures reduce energy consumption in AVs.

Chapter 3

Split-Architecture Computing for Energy-Efficient End-to-End Control

3.1 Introduction and Related Work

The core task of an AV is to perceive the state of the road and safely control the vehicle in place of a human driver. The difficulty in achieving this goal lies in the fact that road scenarios can be highly complex and dynamic, presenting a wide range of potential challenges and obstacles (e.g., rain, snow, construction zones, animals, etc.). To address this challenge, modern AV algorithms rely heavily on (i) large deep learning (DL) models to capture this high degree of complexity and (ii) high-performance edge hardware to reduce processing latency and ensure passenger safety at higher speeds.

As a result, AVs require significant computational power to operate reliably and safely in the real world. However, as AV computing capabilities have scaled up, so have their power and energy requirements. As discussed in Chapter 1, the combined computational and thermal loads induced by modern AV SoCs can reduce an AV's driving range by up to 11.5%

[5], which is especially detrimental for electric vehicles due to their limited range and long recharge times.

Chapter 1.2 also discusses limitations of existing energy saving approaches, such as application-specific hardware design, cloud/fog server offloading, or model simplification/pruning. Of these methods, the only approach that enables computation flexibility without costly hardware or algorithmic changes is offloading. Unfortunately, current offloading approaches have significant scalability and latency issues, as will be discussed in the next paragraph. In contrast, we propose a cloud server offloading methodology that is efficient, safe, and practical for current networking infrastructure.

A naïve solution to the problem of edge energy consumption is to offload self-driving tasks to a cloud server or a Mobile Edge Computing (MEC) server [98]. These 'direct offloading' approaches involve sending images or sensor inputs directly to the server, which processes the data before returning the desired control outputs to the vehicle. However, the real-time latency constraints of autonomous driving and the limitations of current wireless network infrastructure significantly impact this solution's feasibility; to drive and react effectively, AVs must be able to process each input within 100 milliseconds [5]. This bound comes from the fact that the fastest attainable reaction by a human when driving falls within the range of 100-150 ms, meaning that for efficient AV navigation, AVs need to at least perform at the same level as the human driver counterpart. Also, spinning sensors on AVs such as lidar and radar typically have a sensing frequency between 4-20 Hz [19, 18], so AVs should be able to react to each new input sample. Additionally, most real-world AVs, such as those from Tesla [10], Baidu Apollo [11], and Argo AI [99], use multiple high-definition cameras and sensors and would require very high network bandwidths to offload data within the latency constraints. In some cases, the energy needed to transmit and receive data from the cloud server can even *exceed* the energy consumed by edge-only processing. Together, these factors make direct offloading infeasible in most real-world autonomous driving scenarios. Currently, most of the literature has proposed solving this problem by improving network robustness and throughput via solutions such as 5G C-V2X [100] and WAVE [101], or even by placing sensors on the roads themselves [32]. However, implementing these solutions would require significant investments in the networking infrastructure to become realistically feasible.

Several works have proposed methods for offloading some or all AV tasks. For example, [102] proposed a technique for reducing AV processing latency by offloading sub-tasks of LiDAR SLAM to the cloud depending on network conditions. Although they demonstrate good performance, their approach is limited since it only considers LiDAR data, which is significantly smaller than camera data. Additionally, they developed a distributed SLAM algorithm that allowed task-level parallelism; this sort of optimization will need to be applied for every part of a modular AV pipeline and may not be applicable in some areas. In another work, [103] proposed an offloading strategy where computations are executed on either an MEC server or a cloud server depending on network conditions. However, their method requires all sensor input data and internal state information to be sent to the server for processing. Since they only evaluated a micro-car transmitting IMU data (position, velocity, yaw), their approach is not scalable to real-world AVs that would need to offload multiple high-definition camera inputs. The work in [104] proposes a hierarchical approach for offloading in which AVs can offload to road-side units (RSUs) when MEC servers are overloaded, but this work does not consider network bandwidth constraints. Moreover, none of these works [102, 103, 104] considered edge energy consumption in their evaluation, which significantly constrains direct offloading approaches. The authors in [105] evaluated the energy consumption for offloading to MEC servers; however, they do not assess this approach's practicality for large upload data sizes, which are typical for AVs with multiple high-resolution input cameras. In summary, the problem of offloading large data sizes while meeting latency and energy constraints on current network infrastructure is exceedingly challenging and is currently unsolved by existing methods.

3.1.1 Research Challenges

For efficient AV offloading, the following key research challenges need to be addressed:

1. Offloading AV tasks without exceeding safety-critical latency constraints or increasing AV energy consumption.
2. Adapting AV deep learning architectures to support dynamic offloading depending on the corresponding network conditions.
3. Developing a technique efficient enough to meet latency constraints with data from multiple high-definition camera inputs on current industry-standard AV hardware.
4. Producing a cost-efficient, safe solution that can operate within the constraints of current networking infrastructure.

Instead of altering the AV hardware or the communication network infrastructure, we propose **SAGE**: a methodology to significantly reduce the size of the data transmitted over the network and enable efficient computation offloading. By introducing a *bottleneck* layer near the beginning of end-to-end DL control models, the size of the data uploaded to the cloud server is reduced significantly, allowing a large portion of the model computation to be offloaded to the server even at low network bandwidths. This benefit is especially valuable in multi-camera offloading due to the significant bandwidth requirements and edge energy consumption of multi-camera models. Furthermore, it was shown in [106] that, with a particular training strategy, the model's performance after introducing the *bottleneck* remains nearly the same.

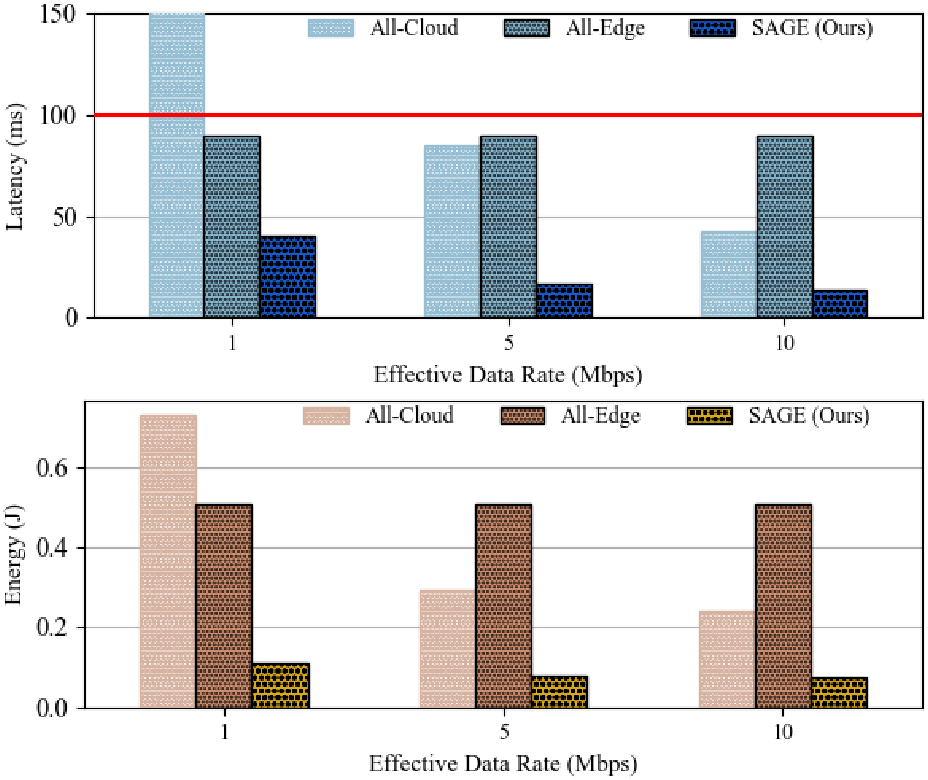


Figure 3.1: A comparative analysis demonstrating the of the overall latency *(top)* and energy consumption *(bottom)* for the *All-Cloud*, *All-Edge*, and *SAGE (Ours)* execution strategies given three typical effective data rate values attainable through a 4G LTE connection. The red line indicates the AV processing latency deadline of 100 ms.

3.1.2 Motivational Example

We provide a brief example to demonstrate the merit of our approach in Figure 3.1. Here, we compare three possible execution strategies for an end-to-end AV control model: executing locally on the edge (*All-Edge*), offloading the entirety of execution to the cloud (*All-Cloud*), and our proposed split approach (SAGE). Our analysis is conducted at three distinct data rate values typical for 4G LTE connections, and the evaluations are performed on a Jetson TX2 for a ResNet-50 model [107] adapted for end-to-end AV control. In terms of latency, it is clear that the *All-Cloud* approach is impractical at low data rate values as it fails to meet the 100 ms processing latency constraint of the AV. On the other hand, performing all the processing locally in the *All-Edge* approach keeps the latency unaffected by the state of the network. However, the downside is that the edge device is fully operational and consumes sizeable amounts of power for longer periods, leading to higher energy consumption in theory than the *All-Cloud* approach at the more favorable data rates. SAGE offers to leverage the best of these both approaches. In brief, SAGE entails replacing an early computational block from the model architecture with a more efficient encoder-decoder-like structure. Then, this modified architecture is divided between the edge and cloud at the encoder output. The encoder, acting as a *bottleneck*, projects the input data into a low-dimensional representation that is more suited to be transmitted to the cloud over the wireless medium. On the other hand, the decoder component is situated as part of the cloud to receive the encoder's output data and map it into a representation analogous to the output of the original computational block from the unaltered model architecture. This structural modification results in significantly lower: (i) local execution latency than the *All-Edge*, and (ii) transmission latency than the *All-Cloud*. Moreover, these improvements are reflected in the energy consumption as the edge device is only required to perform computations for a much shorter interval within the 100 ms time window, which is beneficial for the edge device itself in terms of performance efficiency. More details about the proposed SAGE methodology and how resource-efficiency is promoted across the edge and cloud while maintaining the same degree of accuracy shall be described in detail in Section 3.3.

3.1.3 Novel Contributions

Our novel contributions are as follows:

1. We propose **SAGE**: a novel split-network architecture methodology that allows for a significant reduction in the energy consumption of AV on-board processing units by dynamically offloading part of the model's computations to the cloud.

2. We demonstrate that introducing *bottlenecks* into deep end-to-end AV control models

reduces energy consumption significantly with little to no performance loss.

3. We show that SAGE reduces network throughput requirements significantly compared to conventional cloud server offloading techniques, enabling it to meet latency constraints even at low network bandwidths on 3G, 4G LTE, and WiFi[1].

4. We demonstrate that SAGE is scalable to practical AV use cases by evaluating its performance with three high-definition camera inputs, typical for real-world AVs [10, 11, 99, 108].

5. We demonstrate the practicality and feasibility of our technique by evaluating its performance on the Nvidia Jetson TX2, as well as the industry-standard Nvidia Drive PX 2 autonomous driving platform, used in all 2016-2018 Tesla models for their Autopilot system [10].

3.2 System Model

This section aims to provide a generalized model of how an AV edge device may complete processing a task either through local computation or collaboration with a cloud server. Mainly, the modeling comprises the communication and computation costs that the AV edge device would incur until the task is finished. Our model comprises a direct link between a vehicle i, requiring computation for its designated task, and a cloud server j to whom tasks can be offloaded.

[1] We did not evaluate 5G C-V2X and WAVE in this work because these technologies are currently not widespread and require significant infrastructure changes to be viable. Also, comparable real-world power models for 5G are not available yet in the literature. However, SAGE will be scalable to these emerging technologies.

3.2.1 Communication Model

As the AV runtime optimization solution spans multiple levels in the system architectural hierarchy (i.e., edge and cloud), a communication model is needed to identify the cost of transferring data between entities of different levels. These costs can be represented through transmission latency and energy. More formally, the task to be offloaded can be represented as $t_i = \{a_i, b_i, c_i\}$, where a_i, b_i, and c_i correspond to the size of data to be transmitted, size of data to be received back from the server, and the number of CPU cycles required to complete the task, respectively. To estimate the communication overhead, we will need to determine the upload and download data rates, $r^U_{i,j}$ and $r^D_{i,j}$, experienced at vehicle i's edge device when transmitting data to cloud server j. Although the data rate can be determined theoretically through Shannon's law, this resembles an optimistic estimate, not taking into consideration potential errors or packet losses. Instead, we are more interested in the '*effective*' data rates by which we mean the actual data transfer speeds experienced at the edge device when accounting for errors and re-transmissions. These values can be measured at the target device and accordingly, the upload and download latencies can be given as:

$$T^U_{i,j} = \frac{a_i}{r^U_{i,j}}, \qquad T^D_{i,j} = \frac{b_i}{r^D_{i,j}} \tag{3.1}$$

Thus, the total communication overhead encountered by at vehicle i in terms of latency and energy for offloading task execution to computing server j is given by:

$$T^{comm}_{i,j} = T^U_{i,j} + T^D_{i,j} + T^{RTT}_{i,j} \tag{3.2}$$

$$E^{comm}_{i,j} = p^T_i T^U_i + p^R_i T^D_i \tag{3.3}$$

where p^T_i, p^R_i and $T^{RTT}_{i,j}$ represent vehicle i's transmitting power, receiving power, and the round-trip time between vehicle i and server j, respectively.

3.2.2 Computation Model

Assuming that any task requested by vehicle i consists of several sequential sub-tasks, i.e., as in an end-to-end control pipeline or layers in a DL model, let $\mathbb{C}_i = \{c_{i1}, c_{i2}, ..., c_{iK}\}$ denote the set of K clock cycles required to execute each sub-task. Thus, potential execution times (local or remote) and the energy needed to execute sub-task k locally are:

$$T^l_{ik} = \frac{c_{ik}}{f^l_i} \tag{3.4}$$

$$T^r_{ik} = \frac{c_{ik}}{f^r_i} \tag{3.5}$$

$$E^l_{ik} = \vartheta_i c_{ik} \tag{3.6}$$

where f^l_i, f^r_i and ϑ_i represent the operational frequency at vehicle i, operational frequency at the remote server, and a coefficient denoting energy consumed per CPU cycle at vehicle i. However, since offloading some or all sub-tasks is a viable option in this scheme, the total computational latency and energy consumption for vehicle i can be written as:

$$T^{comp}_i = \sum_{k=1}^{k_p} T^l_{ik} + \sum_{k=k_p+1}^{K} T^r_{ik} \tag{3.7}$$

$$E^{comp}_i = \sum_{k=1}^{k_p} E^l_{ik} + E^{idle}_i(t) \tag{3.8}$$

where k_p is the execution partitioning point after which execution is assigned to the remote server, and $E^{idle}_i(t)$ is the energy consumed by vehicle i waiting for the remote server's results as a function of the idle time t. Note that when $k_p = K$, T^{comp}_i reflects the local execution case without any form of offloading as the second summation becomes an empty sum.

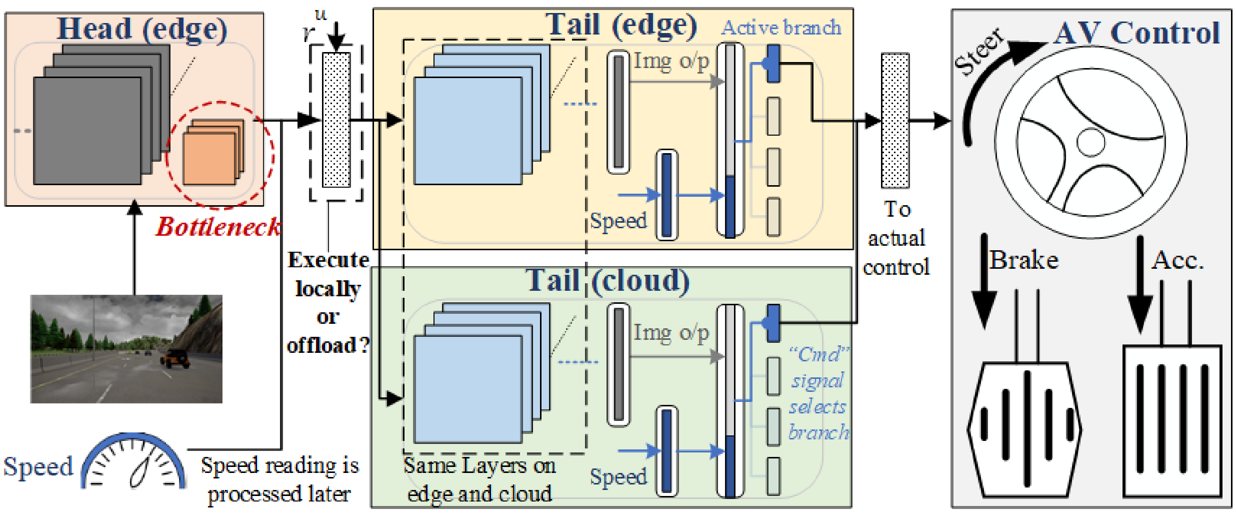


Figure 3.2: An illustration of how systems developed through SAGE support end-to-end AV control. The tail component is replicated on both the edge and the cloud. At runtime, a decision is to be made whether the tail should be executed locally or in the cloud. Final results are applied as inputs to the AV control system.

3.2.3 Problem Formulation

From the previous model derivations, the offloading problem for vehicle i can be formulated as:

$$\begin{aligned} &\min_{k_p} w_i^T(\mathcal{I}(k_p \neq K) \times T_i^{comm} + T_i^{comp}) + w_i^E(\mathcal{I}(k_p \neq K) \times E_i^{comm} + E_i^{comp}) \\ &\quad \text{s.t. } (\mathcal{I}(k_p \neq K) \times T_i^{comm} + T_i^{comp}) <= 100\ ms \end{aligned} \tag{3.9}$$

where w_i^T and $w_i^E \in [0, 1]$ represent user-defined weights associated with the latency and energy metrics, and $\mathcal{I}(k_p \neq K)$ is an indicator function becoming 0 in the case of local execution. As presented earlier, the 100 ms constraint is the window within which the AV must finish its processing task [5]. Note that as k_p varies, so will the values associated with the offloaded task a_i and c_i.

3.3 Methodology

In this section, we present SAGE and discuss its building blocks in detail. Figure 3.2 illustrates how the final system developed through SAGE would support end-to-end AV control. We target end-to-end AV architectures (introduced in Chapter 1.1) because they provide more opportunities for model splitting and allow detailed analysis of each layer's contribution. However, SAGE can also be applied to modular architectures, as discussed in 3.5.2. The implemented DL model is divided into two components: (i) a *head* deployed on the edge and (ii) a *tail* which is replicated across both the edge and cloud. The *head* component contains within its structure a *bottleneck* layer, which represents an optimal offloading point compared to other options. Inputs to the model can come through a camera feed and sensory measurements (e.g., current speed). After the head portion executes at runtime, it is decided whether tail processing should be done locally or be delegated to the cloud depending on current network conditions. The *tail* portion of each DL model contains the bulk of layers and outputs the control values to the AV.

3.3.1 Perception

Much like human beings, perception is concerned with how an AV interprets and understands events occurring in its surrounding environment. To enable perception, AVs are equipped with sensory capabilities to capture representative data from the environment. This data is then processed to extract a comprehensive understanding of the events unfolding around them. Contemporary AVs sense their environment via cameras, LiDAR, or radar equipment [109, 110, 111]. After data acquisition, DL models process the data and estimate the course of action that the AV should take in the following time-step [112]. Without any loss of generality, our evaluations are based on the data-intensive image-based perception from a set of cameras capturing the AV's surroundings. To implement the perception pipeline, we utilize state-of-the-art DL model architectures, which are known to achieve high accuracy on image classification tasks, as baselines. This allows us to leverage these models' abilities to capture fine-grained features from images for processing camera data as part of an end-to-end AV control architecture. Mainly, we consider DenseNet-169 [113], ResNet-34 [107] (used for end-to-end multi-camera AV control in [108]), ResNet-50 [107], and CarlaNet [112] which is implemented specifically as an end-to-end AV control solution.

	DenseNet-169 [113]	ResNet-50 [107]	ResNet-34 [107]	CarlaNet [112]
Perception	98.76%	97.74%	97.36%	59.64%
Imitation Learning (IL)	1.24%	2.26%	2.64%	40.36%
Modified *head* speedup	80.11%	79.97%	67.25%	13.39%
Overall Model speedup	27.01%	41.51%	34.39%	2.65%

Table 3.1: Contribution of perception and IL components in terms of the total processing *(top)*, and how modifying the *head* components in SAGE speeds up model executions *(bottom)*.

3.3.2 Imitation Learning for End-to-End AV Control

Next, the baseline models must be adapted to predict AV control outputs from camera input data. This can be achieved by integrating an Imitation Learning (IL) component at the back-ends of the baseline models to enable them to mimic a human's behavior in regard to a particular task. In this context, the driving algorithm's core objective is to *imitate* the vehicle control outputs (steering angle, brake pedal angle, and accelerator pedal angle) produced by a human driver for a given set of input images [114]. IL models are typically trained via supervised learning, where the goal is to map the input features captured at time-step t to the corresponding human control output values. To effectuate the learning process, a loss function, e.g., Mean Absolute Error (MAE), is used to evaluate the difference between a model's predictions and the ground truth values. Take the baseline ResNet-50 for example, its vanilla network architecture constitutes five main convolutional blocks, representing the main perception component, followed by a final fully-connected layer for image classification tasks. To adapt the model for IL, we replace this fully-connected layer with an IL component developed for end-to-end AV control where the final layer has three separate neurons: one for each control output (steering, accelerator, brake). These outputs are used in both the loss function for MAE computation and controlling the vehicle during deployment. We follow the IL implementation in [112] where firstly, the output from the preceding perception component and the corresponding pre-processed speed measurement at time-step t are concatenated together as the input to the IL component. Next, one of several processing branches is activated based on the driver's command value (e.g., navigation signal). This notion of branching is implemented to associate unique learning features with different driving intentions. For instance, the second branch can only be activated whenever the driver issues the *"Turn right"* navigation signal because this branch's parameters were trained to take actions in anticipation of a right turn, dissimilar to what parameters in other branches learned. The outputs from the active branch are the ones that are directly applied to the AV control system at that particular time step t.

To summarize, a baseline DL-based solution for AV control comprises (i) a perception module, (ii) a speed measurement processing unit, and (iii) an IL back-end. Henceforth, these DL models adapted for IL shall have *"IL-"* preceding their original names, e.g., *IL-ResNet-50.* Moreover, to give an idea of each component's contribution to the overall processing time, The upper part of Table 3.1 shows how perception can be the most computationally-intensive component, especially when utilizing state-of-the-art image classification models. Note that the speed processing unit is executed concurrently with the perception module, which dominates their combined execution time. Thus, our structural modifications target the perception modules to maximize the performance impact.

3.3.3 Structural Alterations for Split Computing

Deploying the AV control algorithm on the edge device is essential for such a mission-critical application. However, dynamically assigning some or all of the processing tasks to a more powerful cloud server if the wireless network conditions are favorable can lead to substantial latency and energy savings on the edge device. As was shown in the motivational example, directly offloading inputs to the cloud can be inefficient at sub-par network conditions: a significant communication overhead can arise from transmitting the raw input images, resulting in a poorer overall performance than that of local execution. Prior work in [115] tries to address this by compressing the input image before transmission, but accuracy degrades significantly. One alternative in [116] proposes scanning each layer within the DL model to identify those which output smaller data sizes than the input as potential data offloading points in a split computing approach. However, this is dependent on each architecture's structure, deeming it ineffective for models that do not shrink data size enough.

A more tractable alternative is modifying the DL model structure by injecting a *bottleneck* amongst the first few layers. This *bottleneck* layer presents an optimal offloading point very early in the model because its structure is designed to output exceedingly small-sized data. This idea is presented in [106, 117, 118], where it is implemented by initially dividing a DL model architecture into two sections: a *head* and a *tail*. The structure of the *tail* remains unchanged. Whereas, a simpler more efficient version of the *head* is constructed to mimic the functionality of the original *head* section. The merits of constructing this new *head* model are twofold. Firstly, the new *head* is structurally more efficient to run than the original *head* providing a local execution speedup, as illustrated in the lower part of Table 3.1. Secondly, the *head* contains the *bottleneck* operating as an encoder-decoder model rigorously transforming its input to lower dimensions (encoder) before raising the dimensionality at its output (decoder), making the encoder serviceable as an efficient data offloading layer. We follow the instructions provided in [106] on how to design a new *head* model with a *bottleneck*

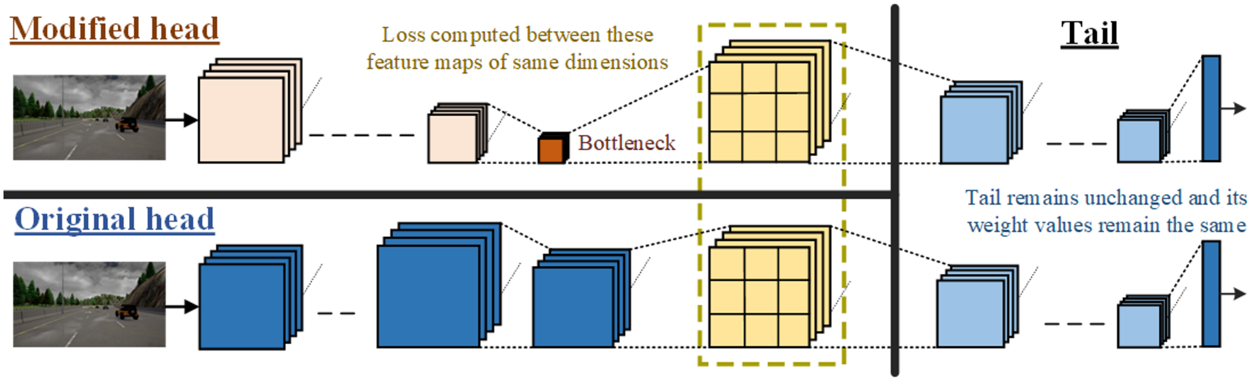

Modified head
Loss computed between these feature maps of same dimensions
Bottleneck
Tail
Original head
Tail remains unchanged and its weight values remain the same

Figure 3.3: The head network distillation process to train the perception component with the *bottleneck*. Note that in the final deployment, the *bottleneck* layer is to be the last layer on the edge device.

from the original *head*. Structurally, both the *bottleneck*'s number of output channels and its preceding layers' complexity should be minimized. However, the modified model's accuracy still needs to be maintained by retraining the new head portion, as discussed in the following subsection. This leads us to place the *bottleneck* earlier in the architecture to minimize local processing overheads, and transmission latency. All in all, offloading at the *bottleneck* is 14× faster than offloading at the input. The overhead for creating a bottleneck layer is analogous to that of creating a small deep learning model manually, which is represented through the human design effort of performing successive refinements in order to attain the desired degree of performance. The main difference is the requirement to have an encoder-decoder structure within the overall architecture to provide the efficient offloading point.

3.3.4 Head Network Distillation (HND)

Knowledge Distillation (KD), presented in [119, 120], has emerged as an effective training technique to render a compressed yet accurate version of a deeper, more complex neural network model. The main reason this technique came about is that shallower neural networks, when trained conventionally, achieve sub-par performance at many tasks compared to deeper networks. Hence, this technique aims to leverage the deeper network as a *teacher* to *distill* its acquired knowledge into the smaller *student* model. Consequently, *student* models trained through KD achieve superior performance relative to their traditionally-trained counterparts [119]. This technique is advantageous when high-performance neural network solutions are needed for edge devices with limited resources. Formally, the *student's* loss function, which is minimized during training, needs to incorporate a distillation component as follows:

$$\mathcal{L}_{student} = \alpha\mathcal{L}_{orig} + (1-\alpha)\mathcal{L}_{KD} \tag{3.10}$$

where $\mathcal{L}_{orig}$ is the conventional loss function using hard labels, whereas $\mathcal{L}_{KD}$ represents the KD loss component, which can be computed using KL divergence, L2 loss, or logits regression [119]. Through providing a control variable α, the effective weight of each loss component can be fine-tuned. This works because the *student* is learning by minimizing the divergence from a vector of the *teacher's* real values, rather than on a single label representation. Hence, the student becomes more capable of capturing the finer details of how the final decision was reached and attempts to learn a simple function to minimize the divergence from this vector of values, thus achieving better generalization.

However, works in [121, 122] discuss how using more complex and accurate *teacher* models makes training through KD for the *student* models more challenging as a result of the capacity mismatch. In these scenarios, more training heuristics are introduced, and more restrictions are imposed on the structures of *student* models. To avoid this in the context of the AV problem, KD is applied between the original and modified *head* components rather than the entirety of models, making them the *teacher* and *student*, respectively. This process entails training the learnable parameters within the modified head model while maintaining the pre-trained tail parameters unchanged from the original model. Consequently, the loss component for the *student-head* model can be computed using the sum of squared difference, as presented in [106]:

$$\mathcal{L}_{KD}(X) = \sum_{x \in X} ||s_h(x) - t_h(x)||^2 \tag{3.11}$$

where s_h and t_h represent the output vectors from the head portions of the *student* and *teacher* on an input x, respectively. For this loss function to be attainable, the final layers in both *head* models must have the same dimensions. Figure 3.3 illustrates the Head Network Distillation (HND) process. Note that although the loss function is computed between the final layers in the *head* modules, in the final deployment, any layers succeeding the *bottleneck* are deployed on the cloud.

3.3.5 Offloading Strategy

After training the modified model using HND and deploying it across the edge and cloud, a runtime strategy must be implemented to determine, for each time step, whether to continue execution at the *bottleneck* or delegate the remaining DL processing tasks to the cloud. The corresponding network conditions, mainly the effective upload data rate: $r_{i,j}^U$, govern this decision. Note that the focus is on $r_{i,j}^U$ because the bulk of the data transmission (tens of kBs) occurs in the uplink, whereas merely the final values (in bytes) are sent through the downlink. So, the task here is to devise a policy based on a data rate threshold r_{th} where:

1. if ($r_{i,j}^U > r_{th}$): the edge device offloads the result of computation at the *bottleneck* to the cloud server where it is processed through the *tail* part of the model before sending the resultant control inputs back to the edge device.

2. *else:* execution proceeds locally at the edge device.

To estimate r_{th}, the 100 ms constraint on AV processing, stated in Equation 3.9, must be considered, where all communication- and computation-related tasks must conclude within that time frame. Moreover, there have to be expected performance gains to justify the offloading decision. Therefore in our formulation, this decision is dependent on whether or not there exist potential energy reductions from offloading. Hence, we can denote r_{th} as:

$$r_{th} = \frac{Upload\ Data\ Size}{100 - (T^{edge}_{head} + T^{cloud}_{tail} + T^{D}_{i,j} + T^{RTT}_{i,j})} \ \ s.t.\ \ (r_{th} > 0)\ and\ (E^{comm}_{i} + E^{edge}_{idle} < E^{edge}_{tail}) \tag{3.12}$$

where T^{edge}_{head} and $T^{RTT}_{i,j}$ represent the edge *head* component's execution time and the round-trip time between the edge and cloud, respectively. The sum of T^{cloud}_{tail} and T^{D} represents the time the edge device is idle waiting for the cloud server to compute and transmit back the control inputs for the AV. The $r_{th} > 0$ restriction guarantees that the sum of the latency estimates in the denominator does not exceed the 100 ms time constraint. Furthermore, the sum of the energy required to offload the data at the *bottleneck* and the idle energy consumption ($E^{comm}_{i} + E^{edge}_{idle}$) must be less than the energy required to execute the tail component of the model (E^{edge}_{tail}) in order to attain a *beneficial* offload. Algorithm 1 demonstrates a runtime algorithm implementing this strategy. We have built this algorithm to promote performance efficiency through offloading whenever the network conditions are benign. Note that ***lines 12-14*** represent a fail-safe mechanism accounting for the network variability *within* a single time window, where it is vital to keep room within the 100 ms time window to invoke local tail execution if the result has not received from the cloud within the expected time limit. This is guaranteed by starting a counter each time window that wakes the edge device to resume execution if the remaining time within the window is equivalent to that of the edge tail model. Also, $T^{RTT}_{i,j}$ in our case is obtained through averaging multiple pings to a remote server, which accounted for $< 10ms$ overhead, however, this value may vary depending on the operational scenarios and the capabilities of network components involved in the communication link. Figure 3.4 illustrates the three possible outcomes from our runtime strategy. It should be noted that, since Algorithm 1 has a computational complexity of $\mathcal{O}(1)$, its execution time is negligible compared to executing the DL models. As such, we excluded its execution time from our calculations.

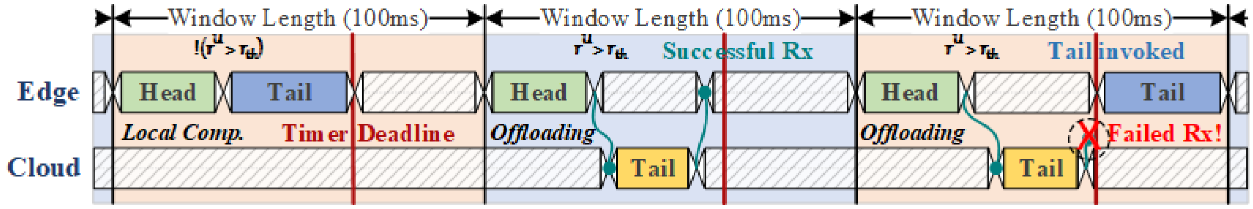


Figure 3.4: Three possible execution scenarios: local execution, successful offloading to the cloud, and unsuccessful offloading to the cloud which entails rolling back to edge computing.

Algorithm 1: Runtime Energy Optimization Algorithm

```
Input: Upload Data Size, $T^{edge}_{head}, T^{cloud}_{tail}, E^{edge}_{idle}, E^{edge}_{tail}$
for each time step t do
    Measure $r^U_{i,j}, r^D_{i,j}$, and $T^{RTT}_{i,j}$          // obtain current network parameters
    Calculate $T^D_{i,j}, r_{th}$, and $E^{comm}_i$          // using current network parameters
    $x = edge\_head()$                                       // execute locally until bottleneck
    if $(r^U > r_{th})$ and $(r_{th} > 0)$ and $(E^{comm}_i + E^{edge}_{idle} < E^{edge}_{tail})$ then
        $Tx\_data(x)$        // transmit bottleneck output to the cloud server
        $Timer \leftarrow reset()$                                       // initialize timer
        $edge\_state \leftarrow idle$                              // edge goes to idle mode
        if $rx\_event$ then
            $edge\_state \leftarrow wakeup$  // edge wakes up to receive server results
            $x = Rx\_data()$
        else if $Timer > (100 - (T^{edge}_{tail} + \epsilon))$ then
            $edge\_state \leftarrow wakeup$         // edge wakes up to execute tail model
            $x = edge\_tail(x)$
    else
        $x = edge\_tail(x)$                                // execute tail model locally
    $Input\_Control(x)$                              // apply control values to the AV
```

3.4 Experiments

3.4.1 Experimental Setup

We evaluate SAGE on two edge devices with significantly different computational capabilities: the Nvidia Jetson TX2 (TX2) and the industry-standard Nvidia DRIVE PX2 AutoChauffeur (PX2). The TX2 is capable of 1.33 TeraFlops (TFLOPS) while operating within a power budget of 15 Watts (W). The more powerful PX2 is designed for real-world autonomous driving use-cases and has been used in vehicles such as the Tesla Model S [10]. It is capable of 8 TFLOPS within a power budget of 80 W. To serve as our cloud server, we used a Windows Desktop with an Nvidia 2080 Super, capable of 11.1 TFLOPS. It should be noted that, in a real-world deployment of SAGE, a more powerful cloud server could be used to increase the benefits of offloading.

In terms of the dataset, we use the CARLA conditional IL dataset from [112]. It contains RGB images in 200x88 resolution and control/sensor values extracted from the CARLA urban driving simulator [63]. We used the image data as well as the steering, accelerator, brake, and navigational command information from the dataset for training and evaluating the accuracy of both our original IL models as well as their bottlenecked counterparts. We implement and train our models in PyTorch to assess the difference in error between the original and *bottlenecked* models. To evaluate the model latency and energy consumption (T_i^{comp} and E_i^{comp} from Equations 3.7 and 3.8), we directly obtained the measurements through the Caffe model timing API for the TX2. For the PX2 and cloud server, we used Nvidia's TensorRT library to compile and optimize the models for the hardware. TensorRT is designed to optimize the model architecture automatically (i.e., optimizing weights, parallelizing computations, combining redundant layers, etc.) to maximize inference performance on a given platform.

In our experiments, we evaluated both 1-camera and 3-camera implementations of our IL models. Our 1-camera experiments aim to demonstrate SAGE for a low-cost AV implementation consisting of either a TX2 or PX2 as the edge device equipped with a single forward-facing camera. This implementation is practical for simple AV tasks such as adaptive cruise control, lane following, etc. Aligning with this goal, we evaluate the energy consumption and feasibility of SAGE with both low-resolution (88x200) and high-resolution (1280x720) camera data.

We also perform 3-camera experiments to demonstrate the feasibility of SAGE on more comprehensive AV hardware platforms. Multi-camera platforms are essential for real-world AV use cases such as urban/highway driving and point-to-point travel. Thus, we evaluated our IL models using three high-definition 1280x720 (720p) camera inputs on the PX2. Here, each model was modified to include three separate perception modules to process data from each camera. The outputs of the perception modules were then concatenated before being processed by the IL module, as was done in [108].

To evaluate the communication power cost needed in Equation 3.3, we use the transmitting and receiving power models derived in [123] for 3G, WiFi, and 4G LTE wireless technologies. Note that 5G energy evaluations are not available since we could not find any 5G-specific real-world power models in the literature as we found for the other technologies. In terms of the computation energy in Equation 3.8, we leverage the onboard sensing circuits within the TX2 board for estimating the execution and idle powers, whereas we use an external power meter for the PX2. We assume no packet losses in our evaluations, and as mentioned, we demonstrate SAGE's feasibility with widespread and currently available network technologies.

3.4.2 Performance Comparison of Original vs. Bottlenecked Models

Recall that the *bottleneck* acts as an encoder whose main purpose is to reduce the output data size to attain an efficient data transmission if needed. This data reduction is mainly achieved through reducing the number of *output channels* at the *bottleneck* layer (3 in the experiments). To give perspective, the *output channels* for any layer in any of the original DL models discussed here before introducing our alterations is 32, meaning that there is an $\approx 10\times$ reduction in output data size at least. To ensure that the introduction of a *bottleneck* into our models does not impact their predictive performance, we evaluated the mean absolute error (MAE) of our models both with and without the bottleneck, shown in Table 3.2. In the cases with the bottleneck, we used HND to train the *head* of the bottlenecked model to mimic the original model's *head*, as described in Section 3.3.4. The results clearly show that the bottlenecked models perform very similar to the original models, with only a slight increase in MAE. For context, an MAE increase of 0.01 corresponds to a 1% increase in error between the model outputs and the outputs provided by the human driver.

Model	Mean Absolute Error (MAE)		
	Steering	Accelerator	Brake
IL-DenseNet-169	0.0177	0.0356	0.0129
IL-DenseNet-169 w/HND	0.0159 (-0.0018)	0.0509 (+0.0153)	0.0195 (+0.0066)
IL-ResNet-34	0.0259	0.0506	0.0199
IL-ResNet-34 w/HND	0.0263 (+0.0004)	0.0545 (+0.0039)	0.0259 (+0.0060)
IL-ResNet-50	0.0260	0.0514	0.0180
IL-ResNet-50 w/HND	0.0266 (+0.0006)	0.0601 (+0.0087)	0.0330 (+0.0150)
IL-CarlaNet	0.0259	0.0546	0.0228
IL-CarlaNet w/HND	0.0204 (-0.0055)	0.0589 (+0.0043)	0.0326 (+0.0098)

Table 3.2: Comparison between the original IL models and their modified counterparts with *bottlenecks* after HND. Values in parentheses are the differences in error between the models.

3.4.3 Power, Energy, and Latency Evaluation on Hardware

From this point onwards, all IL models referred to are with *bottleneck* layers added. In Table 3.3, we compare the power consumption, energy consumption, and latency of different parts of the IL models on each hardware platform. By comparing the end-to-end (E2E) latency of the edge devices with the edge head latency and server tail latency, we see that offloading at the head provides ample time to account for network transmission latency. Furthermore, the table shows a significant energy reduction when processing the head model instead of the entire model end-to-end. These metrics illustrate the feasibility and potential benefits of our model.

Network	Device	Power (W)			Latency (ms)			Energy (J)	
		E2E	Head	Idle	E2E	Head	Tail	E2E	Head
IL-DenseNet-169	Server	–	–	–	–	–	**2.238**	–	–
	TX2	5.446	5.430	1.659	215.543	**8.043**	207.5	1.1740	**0.0437**
	PX2	43.58	47.42	40.23	10.420	**1.112**	9.308	0.4541	**0.0527**
IL-ResNet-34	Server	–	–	–	–	–	**0.572**	–	–
	TX2	5.95	5.221	1.659	65.560	**11.612**	53.948	0.3901	**0.0606**
	PX2	46.99	47.51	40.23	4.534	**0.695**	3.839	0.2131	**0.0330**
IL-ResNet-50	Server	–	–	–	–	–	**0.607**	–	–
	TX2	5.682	5.415	1.659	89.231	**10.432**	78.799	0.5070	**0.0565**
	PX2	46.89	47.17	40.23	7.413	**1.195**	6.218	0.3476	**0.056**4
IL-CarlaNet	Server	–	–	–	–	–	**0.188**	–	–
	TX2	5.391	5.039	1.659	28.795	**8.727**	20.068	0.1552	**0.0440**
	PX2	45.54	46.33	40.23	1.659	**0.593**	1.066	0.0756	**0.0275**

Table 3.3: Hardware performance metrics for processing one 88x200 camera input. E2E = processing the entire model end-to-end on the edge device. Cloud server power/energy is ignored because this is not a constraint.

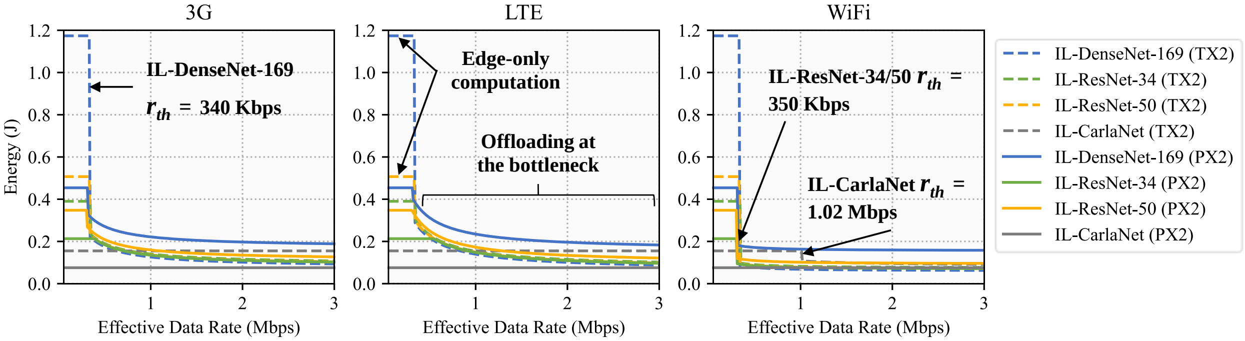


Figure 3.5: Energy consumption of IL models developed through SAGE while processing a single 88x200 camera input at different data rates with 3G, 4G LTE, and WiFi. The transition point in each line occurs at r_{th}, which is when offloading begins at the *bottleneck*. Before this point, the energy consumption for the edge-only processing is $(E^{edge}_{head} + E^{edge}_{tail})$. After this point, the energy consumption is calculated as $(E^{edge}_{head} + E^{comm}_{i} + E^{edge}_{idle})$.

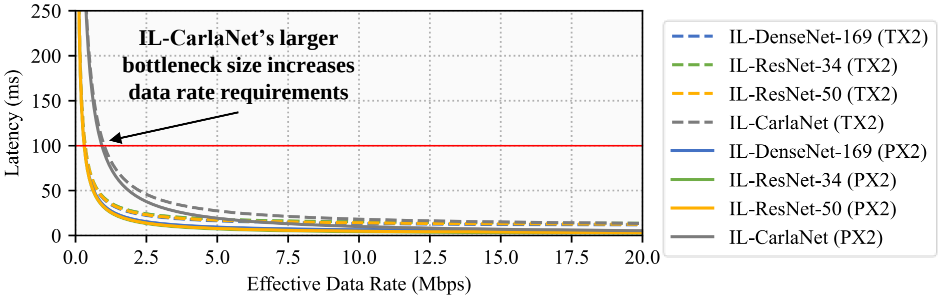


Figure 3.6: End-to-end latency of each model for offloading at the *bottleneck* for an AV with a single 88x200 camera input. The end-to-end latency includes edge head processing latency, wireless network latency, and server processing latency at various network data rates. The red line indicates the 100 ms latency constraint

3.4.4 Offloading Evaluation

Low Resolution

In Figures 3.5 and 3.6, we show results from evaluating models implemented through SAGE with a single 200x88 resolution camera input using the TX2 and the PX2.

Figure 3.5 shows the energy consumption of each IL model with each technology type at different values of effective data rate r^U. Recall that these values are obtained based on the offloading strategy in Section 3.3.5, where it is only feasible to offload when $(r^U_{i,j} > r_{th})$ and

$(E_i^{comm} + E_{idle}^{edge} < E_{tail}^{edge})$. For each model, observe the sharp change in Figure 3.5 at r_{th} where the model switches from edge-only computation to cloud offloading. For IL-DenseNet-169, IL-ResNet-34, and IL-ResNet-50, this switching point occurs at approximately 350-400 Kbps on both the TX2 and PX2.

Although offloading can meet the latency constraint for some r^U values, the energy consumed by the networking components must still be considered. As shown in Figure 3.6, IL-CarlaNet can feasibly offload at 1 Mbps on both devices, but on 3G and 4G LTE, offloading consumes more power than edge-only computation. Thus, we only consider r^u values which are greater than r_{th}, at which offloading *saves* energy on the edge device compared to edge-only processing. For IL-CarlaNet on the TX2, r_{th} is 7.7 Mbps on 3G, 3.62 Mbps on 4G LTE, and 1.02 Mbps on WiFi. This is likely because IL-CarlaNet has a larger data size at the *bottleneck* than the other models, increasing communication latency and energy. Interestingly, on the PX2, IL-CarlaNet's r_{th} is 13.66 Mbps for WiFi and there is no 3G or 4G LTE r_{th} under 100 Mbps for IL-CarlaNet that saves energy compared to edge-only computation. This is likely a result of the data size and the fact that IL-CarlaNet is a very small model and the PX2 has a moderately high idle power consumption (40.23W), meaning that offloading would consume more power than simply running on the edge. Since IL-DenseNet-169, IL-ResNet-34, and IL-ResNet-50, are larger models, there is a clear benefit to offloading. Thus, the r_{th} remains at 320-390 Kbps. The only exception is IL-ResNet-34 using LTE on the PX2, which has an r_{th} of 550 Kbps, likely due to the efficiency of the PX2 compared to its idle power consumption and the network latency at this data rate.

Overall, when offloading at the r_{th} for each model and technology, the TX2 and PX2 consume an average of **49.78%** and **22.48%** less energy, respectively, compared to edge-only computation. Interestingly, when running edge-only, the PX2 consumes half as much energy as the TX2; however, when both offload at r_{th}, the PX2 consumes $\approx 25\%$ more energy than the TX2. This is likely because the network latency outweighs the efficiency benefit of the

PX2 at these low throughputs. Regardless, both devices significantly reduce edge energy consumption by offloading.

For all models except IL-CarlaNet, the r_{th} is well within the operating range for all three network technology types. Figure 3.6 clearly shows that all models can meet the deadline of 100 ms with network throughputs as low as 320 Kbps. Above 15 Mbps, the benefit of higher data rates is minimal for this data size.

High Resolution

The previous experiment demonstrated that our approach is feasible and has significant benefits for low-resolution camera data. However, real-world AVs use high-definition cameras to improve perception performance and safety [11, 10, 99]. To emulate this application, we evaluate SAGE on camera data with a 1280x720 (720p) resolution, the resolution used for Tesla Autopilot 2.0 systems. We only assessed the PX2 on this application since it is infeasible for the TX2 to meet the deadline of 100 ms with this input size even when running on the edge only. The results of this experiment are shown in Figures 3.7 and 3.8.

The larger input image size increases model sizes and data sizes at the *bottleneck* (59× larger for IL-CarlaNet and 47× larger for all other models), increasing the edge processing latency, communication latency, and energy consumption significantly. This change is reflected in the figures, as IL-ResNet-34 and IL-ResNet-50 have an r_{th} of $\approx$ 16.5 Mbps. This data rate is well within the normal operating ranges of 4G LTE and WiFi connections. IL-DenseNet-169 has r_{th} values of 30.53 Mbps and 16.65 Mbps on 4G LTE and WiFi, respectively, but does not have a practical r_{th} under 100 Mbps for 3G. This is likely because 3G consumes significantly more energy to upload data than 4G LTE and WiFi. Also, TensorRT better optimized the IL-DenseNet-169 model since it consists of a large number of relatively small layers, reducing its energy consumption significantly compared to IL-ResNet-34 and IL-ResNet-50.

This reduction decreases the potential benefits of offloading in this case. Compared to edge-only processing, offloading the models at r_{th} with 3G, 4G LTE, and WiFi reduces edge energy consumption by **48.54%**, **41.96%**, and **50.72%**, respectively. With high-resolution data, the energy consumption benefit is more than double that of offloading low-resolution data, indicating that offloading is more beneficial for large, demanding edge models.

Since the data size at the IL-CarlaNet model's *bottleneck* is 3.86× larger than that of other models, it requires a higher throughput (57.8 Mbps) than the other models to meet the deadline. Also, IL-CarlaNet's small model size reduces its edge energy consumption, meaning that the communication energy consumption and idle power consumption could outweigh any potential savings. We found no r_{th} below 100 Mbps for any networking technology that reduces the energy consumption of IL-CarlaNet below that of edge-only processing.

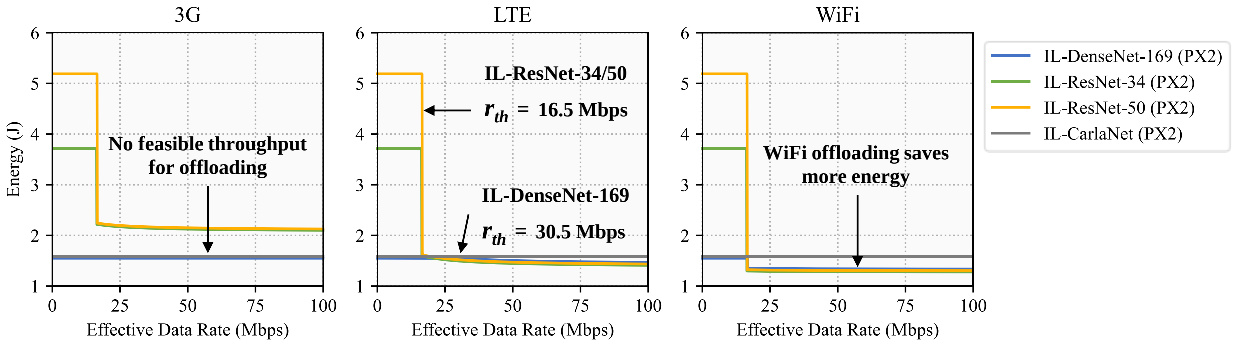


Figure 3.7: Energy consumption of each model for processing a single 1280x720 (720p) camera input

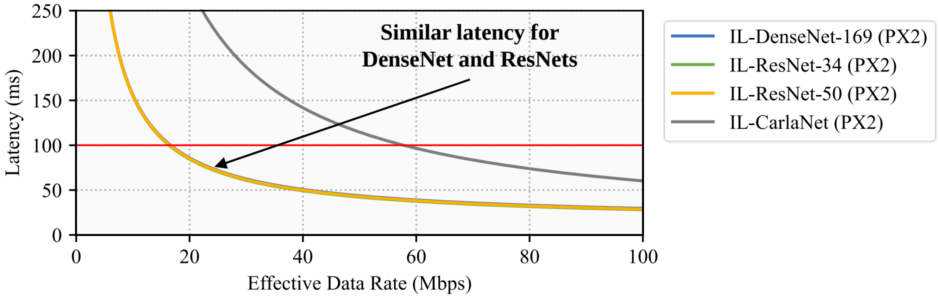


Figure 3.8: End-to-end latency of each model for offloading at the *bottleneck* at different network data rates for an AV with a single 1280x720 (720p) camera input.

3.4.5 Multi-Camera Evaluation

State-of-the-art AVs use multiple high-definition cameras to capture more information about the vehicle's surroundings to improve decision-making, control, and safety [108, 10, 11, 99]. This problem is highly demanding in terms of energy consumption and network connectivity since the latency constraint remains the same at 100 ms despite the significant increase in input and model size. To evaluate SAGE on this application, we provide three 720p camera inputs to our models.

We adapt our models for this task by replicating the original 720p perception pipelines to form three parallel perception pipelines (one for each camera input). The outputs of these pipelines are then concatenated and passed to the IL portion of each model. Consequently, each of the parallel perception pipelines contains one *bottleneck* layer from which data can be offloaded. During offloading, we assume the data at all three *bottlenecks* are sent to the cloud simultaneously. To reduce the maximum throughput requirement in this application, we quantize the values at the *bottleneck* from 32-bit precision to either 16-bit or 8-bit precision before transmission. We tested IL-DenseNet-169 with quantizations of 16-bits and 8-bits at the *bottleneck* layer and found that the average difference in MAE compared to the original is just 1.6×10^{-10}, which is imperceptible. Thus, with 16-bit and 8-bit quantization, we reduce our throughput requirements by 50% and 75%, respectively, while having a negligible effect on performance. Once again, we only evaluate the PX2 in this application since the TX2 cannot meet the deadline of 100 ms with the 3-camera models.

In this application, all-cloud offloading approaches are entirely infeasible. Given that the input data size (three 720p images) is 8.29 MB total, they would require a minimum throughput of 664 Mbps to meet the 100 ms deadline. In contrast, the data size offloaded by our model with 16-bit *bottleneck* quantization is only 264 KB (31× smaller); with 8-bit quantization, this drops to 132 KB (62× smaller). In our experiments, we find that our approach is feasible at throughputs easily achievable by WiFi and 4G LTE. Our experimental results are shown in Figures 3.9 and 3.10.

As shown in Figures 3.10 and 3.9, with 16-bit quantization, IL-DenseNet-169, IL-ResNet-34, and IL-ResNet-50 can all offload at r_{th} values of 51.57 Mbps, 37.98 Mbps, and 39.05 Mbps, respectively. With 8-bit quantization, these r_{th} values drop to 25.79 Mbps, 18.99 Mbps, and 19.53 Mbps, respectively. With 8-bit quantization, most 4G LTE and WiFi connections can easily support the r_{th} data rates. Regarding 16-bit quantization, good quality 4G LTE and most WiFi connections should be able to support the r_{th} data rates [124]. On 4G LTE and WiFi, these models consume **52.67%** and **50.40%** less energy, respectively, by offloading at their r_{th} throughputs. The energy reduction is much more significant for IL-ResNet-34 and IL-ResNet-50 than IL-DenseNet-169, which we again attribute to TensorRT's model optimizations. It should be noted that, during offloading, all models appear to have very similar energy consumption. Practically, this means that an AV can run much larger models (e.g., use IL-ResNet-50 instead of IL-ResNet-34) without much difference in energy consumption provided a network connection with a data rate greater than r_{th} is available most of the time.

Once again, there is little benefit for offloading IL-CarlaNet due to the larger data size at the *bottleneck* (1.01 MB) and the relatively low energy consumption of the model running on the edge. With 16-bit quantization, IL-CarlaNet only saves energy on WiFi at a data rate above 99.51 Mbps. However, with 8-bit quantization, offloading becomes feasible for both 4G LTE and WiFi at 49.76 Mbps. Since IL-CarlaNet is a relatively small model, it may be better to run it on the edge device most of the time and only offload on WiFi when network throughput is high.

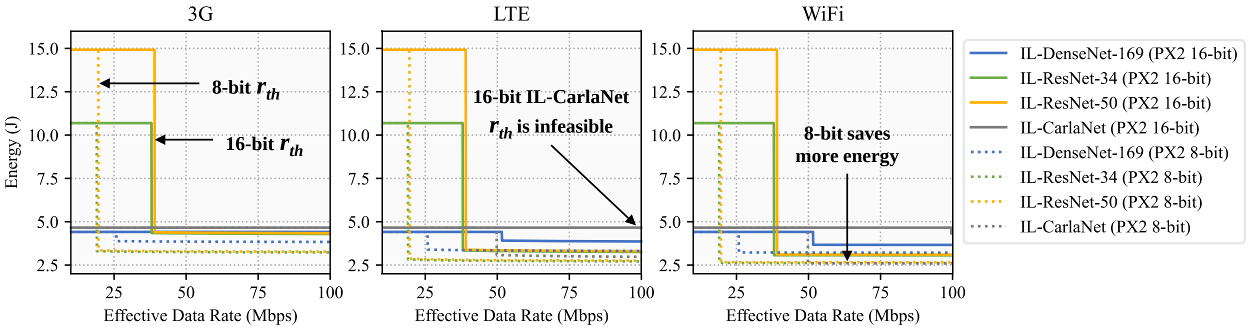


Figure 3.9: Energy consumption of each model for processing three 1280x720 (720p) camera inputs. Results are shown for both 16-bit quantization and 8-bit quantization at the *bottleneck*.

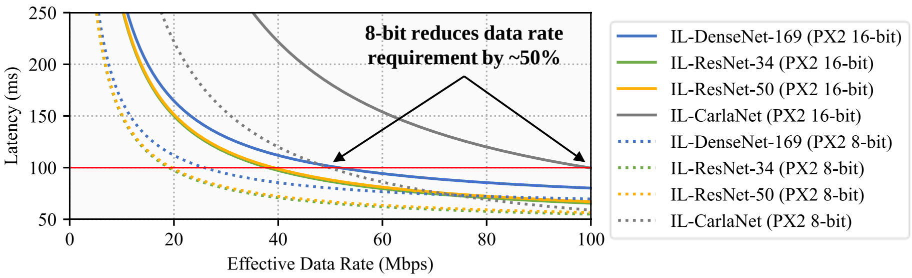


Figure 3.10: End-to-end latency of each model for offloading at the *bottleneck* at different network data rates for an AV with three 1280x720 (720p) camera inputs. Results are shown for both 16-bit quantization and 8-bit quantization at the *bottleneck*.

3.5 Discussion

In this section, we discuss our key findings from our experiments as well as the limitations, feasibility, and cost of SAGE. We also discuss future research directions.

3.5.1 Overall Findings

We found that SAGE was feasible for most IL models for all hardware configurations. By offloading at r_{th}, SAGE reduced edge device energy consumption by **36.05%** with one low resolution camera, **47.07%** with one high-resolution camera, and **55.66%** with three high-resolution cameras. More energy could be saved by offloading at throughputs higher than r_{th} when possible. Additionally, our results indicate that SAGE saves more energy by offloading when input data sizes are larger (i.e., when using more cameras or higher resolutions). SAGE also reduces upload data size by **96.81%** and **98.40%** with 16-bit and 8-bit quantization, respectively, compared to directly offloading three 720p camera inputs. Besides, we found that our introduction of *bottleneck* layers only increased mean error by $\approx 1\%$ and quantization had a negligible effect on error, meaning that SAGE could be scaled to even higher camera resolutions easily.

3.5.2 Limitations

In our experiments, we found that our offloading methodology was not particularly effective for IL-CarlaNet. With low-resolution data, it required a significantly higher r_{th} to provide a benefit than the other models; with high-resolution data and multiple cameras, there was no r_{th} below 100 Mbps that reduced energy consumption. In its current form, SAGE may not present useful offloading for small models and models with a proportionally large *bottleneck* size due to the increased energy cost of transmitting and receiving data compared to just running the entire model on the edge.

Additionally, although the 100 ms represents a reasonable worst-case bound, the current industry standard for real-time video processing is 30 frames/second, meaning that practically, the bound for completing the AV prediction task can be even tighter reaching ≈ 33 ms. From our experimental analysis, SAGE can meet this constraint when offloading the quantized version of the single full HD image data transformation. However, it fails to satisfy this requirement in the case of 3 HD camera inputs. Thus, experimentation with respect to AV industry-standard hardware and 5G wireless technology can provide a fair assessment of SAGE's capability to meet these tighter bounds.

Although our methodology has shown promise in terms of improving the overall performance efficiency, several other factors can impact the extent of this improvement given some real-world situations. It is possible that channel contention between users, packet loss, and channel coherence issues related to vehicle speed and environmental conditions could limit the benefits of our methodology. These effects are difficult to simulate accurately, so real-world experiments are still needed to gauge the energy savings offered by our methodology in these situations.

Lastly, we did not evaluate our approach on modular pipelines. However, since modular pipelines' perception modules generate the most latency [5], SAGE could be directly applied to these modules to achieve similar energy benefits. AV hardware platforms also handle other tasks such as route planning and user interfaces, but these applications constitute a minute part of the overall AV driving system. [5] has shown that the object detection, tracking, and localization modules (i.e., components of the modular version of the perception pipeline) comprise over 98% of the total computation, consuming 1.99 J per input. This proportion is very similar to the results we show in Table 3.1. Based on our energy savings with 3-camera offloading, if we introduce a *bottleneck* to the object detection module and offload the remaining modules to the cloud, we could reduce energy consumption from 1.99 J to 0.896 J, a savings of 55%.

3.5.3 Practicality and Cost

Since SAGE does not require any hardware modifications to the AV or network infrastructure, it is much more cost-efficient and flexible than other solutions such as ASIC design or 5G C-V2X/WAVE installation. The only added costs are those associated with hosting a cloud server to run the offloaded models. However, we demonstrated SAGE's feasibility with a Desktop PC as the cloud server, so hosting similar hardware in the cloud would likely be inexpensive. These costs could even be passed on to consumers, where a vehicle owner could elect to extend their AV driving range by paying for an offloading service as proposed in [104]. Compared to direct offloading, SAGE has significantly lower throughput requirements, making it much more practical for real-world deployment with the current networking infrastructure.

3.5.4 Future Work

In this work, we demonstrated the performance benefits attainable through the SAGE methodology over two NVIDIA hardware platforms, JETSON TX2 and DRIVE PX2. Although our approach is platform-agnostic, we intend to apply SAGE in our future works on different target hardware with different capabilities, like the high performance inference Neural Processing Units (NPUs) developed by ARM [125]. To ensure that our methodology does not introduce additional safety risks, it would also be prudent to evaluate each model on closed-loop evaluations in future work, such as judging each model's success rate at driving point-to-point in a simulator as in other works [112, 126, 127, 63]. Moreover, even though we demonstrated the merit of SAGE using the current prevalent network technologies, this research area is still relatively new, and problems such as energy optimization with multiple servers, modular AV architectures, and 5G networks remain unstudied. For example, SAGE can be adapted to address a multi-MEC server problem context. In this case, the action-space would expand from the AVs' perspective, for they would not only need to make an offloading decision each time step, but also identify which server should be selected for data transfer and task delegation. This would also entail additional dynamic factors to be considered, such as each server's load. Hence, a more sophisticated approach, like reinforcement learning [128], would need to be applied to solve the problem each time-step, in which previous connection experiences with the various servers could be leveraged through an in-place policy to guide the MEC server selection. These problems are left to be addressed in future works.

3.6 Summary

Designing AV control algorithms that are both safe and energy-efficient is a complex challenge that cannot be practically solved using simple direct offloading strategies. In this chapter, we propose SAGE: a methodology for splitting the computation of IL end-to-end control models between the edge and the cloud while minimizing network throughput requirements by adding *bottleneck* layers to the models. We evaluate SAGE on both large and small IL models and show that adding *bottleneck* layers only results in a minor performance impact. Our experiments demonstrate that SAGE reduces the edge energy consumption of IL end-to-end control algorithms with both low-resolution and high-resolution camera data by **36.13%** and **47.07%**, respectively. Additionally, we show that SAGE is scalable to AVs that use three high-definition camera inputs, reducing energy consumption by **55.66%**, and can be practically implemented using current state-of-the-art AV hardware (PX2) and networking infrastructure (3G, 4G LTE, and WiFi). On all three applications, we demonstrate that the IL models can be offloaded at effective data rates that are well within the constraints of current network infrastructure while still meeting AV latency deadlines. We also find that the throughput requirements for offloading reduce by 50% and 75% when quantizing the *bottleneck* output to 16-bits and 8-bits, respectively, with a negligible change in model performance. Overall, we show that SAGE is practical for real-world, end-to-end control applications and can significantly curtail AV energy consumption.

This chapter demonstrated how *dynamic routing* networks conditioned on network conditions can enable significant energy savings on the edge. In the following chapter, we study how *dynamic width* networks conditioned on either domain-knowledge context or abstract global state context (modeled from sensor data) can enable energy-efficient, high-fidelity perception across diverse driving scenarios.

Chapter 4

Context-Aware Dynamic Architectures for Energy-Efficient Sensor Fusion

4.1 Introduction

As discussed in Chapter 1.2 and Chapter 3, the computational demands of heterogeneous sensors and large DL models significantly increase the hardware requirements of AS and limit operating range. Since some of the largest energy consumers in AVs are the sensing and perception modules, energy-efficient perception and sensor fusion can help alleviate these computational demands and improve range.

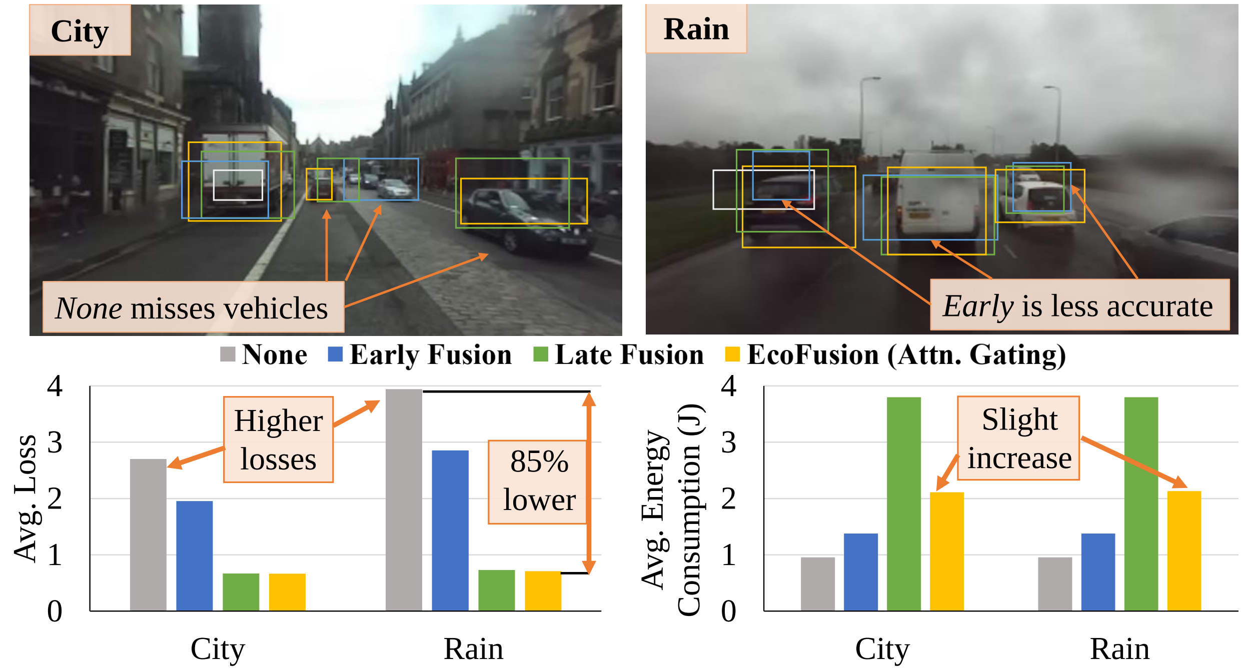


Figure 4.1: Performance and energy comparison for various AV perception sensor fusion methods in city and rainy driving.

4.1.1 Research Challenges

Several works have proposed efficient sensor-fusion approaches that attempt to combine multiple sensing modalities to achieve good perception performance with less energy than conventional fusion [129, 52, 130]. However, these approaches are also limited because they use statically designed fusion algorithms (e.g., early or late fusion) that can lack robustness in difficult driving scenes [131]. Figure 4.1 illustrates the trade-off between performance and energy between different sensor fusion methods for two contexts: city and rain. *None* refers to using a single sensor with no fusion, *early fusion* combines raw sensor data before processing, and *late fusion* processes each sensor separately before fusing the final outputs. As shown, *no fusion* consumes the least energy but also performs the worst, *late fusion* performs much better but uses almost 3x more energy, and *early fusion* is energy efficient but performs poorly in difficult driving scenarios.

In summary, our key research challenges include: (i) perceiving the environment accurately in difficult contexts, (ii) reducing the energy consumption of AV perception systems, and (iii) adapting the perception model to the current context to minimize energy consumption without compromising perception performance.

4.1.2 Novel Contributions

Humans intuitively leverage contextual information about the driving scene (*e.g.*, weather, lighting, road type, high-level visual features) to adjust their decisions and focus while driving. Similarly, contextual information can inform AV perception and enable the fusion algorithm to adapt to different scenarios. To address the above research challenges, we propose **EcoFusion**: an energy-aware sensor fusion approach that uses context to dynamically switch between different sensor combinations and fusion locations. Our approach can reduce energy consumption without degrading perception performance in comparison to both early and late fusion methods. As shown in Figure 4.1, our approach (shown in gold) achieves higher performance than other fusion methods while significantly reducing energy consumption.

The key contributions of this chapter are as follows:

1. We propose EcoFusion: an energy-aware sensor fusion approach that uses context to adapt the fusion method and reduce energy consumption without affecting perception performance.

2. We propose novel gating strategies that can identify the context and use it to dynamically adjust the model architecture as part of a joint optimization between energy consumption and model performance.

3. We benchmark the hardware performance of EcoFusion on the industry-standard Nvidia Drive PX2 autonomous driving platform.

4. We present an in-depth analysis of the performance of each sensing modality in a range of difficult driving contexts.

4.2 Related Work

In past years, research on energy-efficient AVs has focused mainly on reducing the energy needs for locomotion and actuation. However, due to the rise in DL perception algorithms and the computational requirements of modern AVs, minimizing the energy consumption of AV E/E systems is becoming a core problem [132, 133]. Authors in [130] focus on improving computational efficiency through algorithmic changes for a camera-lidar AV platform while using knowledge-based network pruning in their DL model. Selectively fusing sensors, as done in [134], also has potential benefits to save computational energy on AVs. Distinct from these methods, EcoFusion utilizes the context of the environment to enable further energy optimization for AVs. Studies have demonstrated the value of context identification, such as in [52], where authors propose altering the power levels and operating state of an AV lidar sensor depending on the environmental factors, such as the vehicle's speed, to improve perception efficiency. Likewise, [129] proposes adjusting the sensing frequency for indoor robot localization according to environmental dynamics. However, these approaches are limited as they rely on statically designed context-based rules, whereas EcoFusion employs a self-adaptive design to learn the context of the environment dynamically. This approach to context-modeling improved perception robustness in [131], however this work did not consider the energy consumption of different model configurations.

Trade-offs between the energy and performance of deep neural networks (DNNs), like those used in AV perception, have been studied in several prior works. [135] improves the computational efficiency of DNNs for classification by using component-specialization during training and component-selection during inference. [136] presents a structure simplification procedure that removes redundant neurons within DNNs. [137] performs incremental training with DNNs to consider energy-accuracy trade-offs at run-time. Unlike EcoFusion, these works are only applied to classification using a single input modality and do not incorporate context. Additionally, we tackle the complex, cross-domain problem of AV energy optimization with our dynamic sensor fusion architecture, and present experiments involving real AV hardware.

4.3 Problem Formulation

Here we detail the formulation for AV object detection and the joint energy-performance optimization implemented in our work.

4.3.1 Sensor Fusion for Object Detection

For each input sample, the goal of an object detector ϕ is to utilize the set of sensor measurements in the sample, $\mathbf{X}$, to accurately detect the objects in the scene, $\mathbf{Y}$:

$$\mathbf{Y} = \phi(\mathbf{X}),\ \text{where}\ \mathbf{Y} = \{\mathbf{Y}^i_{class}, \mathbf{Y}^i_{reg}\}_{i=1\dots d} \tag{4.1}$$

where d is the number of objects in the sample. ϕ can be implemented via conventional sensor fusion techniques, an ML/DL model, or an ensemble of ML/DL models. The targets for object i in the sample are defined as follows:

$$\mathbf{Y}^i_{class} \in \{c_1, c_2, c_3, \dots\},\ \mathbf{Y}^i_{reg} = [\mu_1, \nu_1, \mu_2, \nu_2] \in \mathbb{R}^2 \tag{4.2}$$

where $\mathbf{Y}^i_{class}$ represents the class of the object (e.g., c_1: car, c_2: truck, c_3: pedestrian) from a set of defined object classes, and $\mathbf{Y}^i_{reg}$ represents the 2D bounding box coordinates of the object in reference to the coordinate frame of the sample. We denote the model's estimate of $\mathbf{Y}$ as $\hat{\mathbf{Y}}$.

Since $\mathbf{X}$ represents data from multiple heterogeneous sensing modalities, sensor fusion can be used to fuse the data to provide a better estimate of $\mathbf{Y}$. In early fusion, the raw sensor inputs are fused before being passed through the object detector as follows:

$$\hat{\mathbf{Y}} = \phi(\psi(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_s)) \tag{4.3}$$

where ψ represents the function for fusing the different inputs. In contrast, *late fusion*, involves fusing the outputs of an ensemble of sensor-specific object detectors as follows:

$$\hat{\mathbf{Y}}_1,\, \hat{\mathbf{Y}}_2,\, \ldots,\, \hat{\mathbf{Y}}_s = \phi_1(\mathbf{X}_1),\, \phi_2(\mathbf{X}_2),\, \ldots,\, \phi_s(\mathbf{X}_s) \tag{4.4}$$

$$\hat{\mathbf{Y}} = \phi(\hat{\mathbf{Y}}_1, \hat{\mathbf{Y}}_2, \ldots, \hat{\mathbf{Y}}_s) \tag{4.5}$$

4.3.2 Energy Modeling

In this work, we aim to jointly optimize the energy consumption and performance of the perception system of an AV. To enable this optimization, we use real-world measurements from three different sensors to model the energy consumption of various object detectors ϕ on the industry-standard Nvidia Drive PX2 autonomous driving hardware platform, depicted in Figure 4.2. For a given object detector implementation ϕ and fixed-size input $\mathbf{X}$, we model energy consumption $\mathbf{E}$ as follows:

$$\mathbf{E}(\phi, \mathbf{X}) = \mathbf{P}(\phi, \mathbf{X}) * t(\phi, \mathbf{X}) \tag{4.6}$$

where $t(\phi, \mathbf{X})$ represents the processing latency in seconds, and $\mathbf{P}(\phi, \mathbf{X})$ represents the hardware power consumption in Watts of running input $\mathbf{X}$ through ϕ as measured on the hardware. We measured the PX2's average power consumption under load as 45.4 Watts. Assuming X has a fixed size, we calculate $E(\phi)$ for all $\phi \in \Phi$ offline. Next, we use this energy calculation within a joint optimization framework.

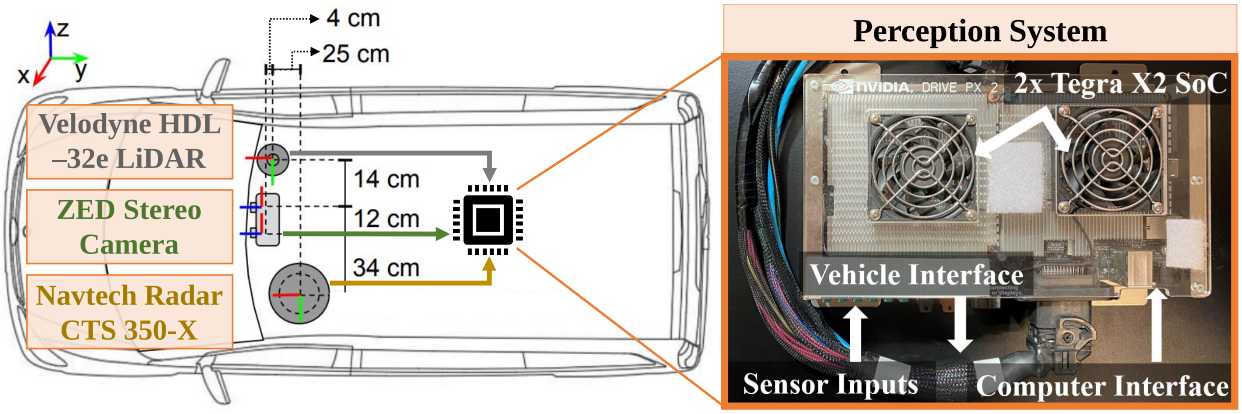


Figure 4.2: Sensor diagram [1] with our Nvidia Drive PX2.

4.3.3 Joint Energy-Performance Optimization

We formulate our optimization as a joint minimization problem between energy consumption and model loss. We denote the list of all object detector configurations as Φ. For each configuration ϕ in Φ, we use a model to predict the loss after the outputs of ϕ are fused via late fusion, denoted $L_f(\phi)$. The loss is defined as the combined regression and classification loss (using smooth L1 loss and cross-entropy loss, respectively) between the ground-truth $\mathbf{Y}$ and the $\hat{\mathbf{Y}}$ predicted by the model as defined in [138]. Then, the minimum fusion loss configuration ϕ' is identified. We also define the function ρ, which determines the set of ϕs that have a fusion loss within γ of ϕ'. This set Φ^* is defined as follows:

$$\Phi^* = \rho(L_f(\Phi), \gamma) = \{\phi \in \Phi \text{ s.t. } L_f(\phi) - L_f(\phi') \leq L_f(\phi') + \gamma\} \tag{4.7}$$

where γ is the maximum allowable difference in loss between any ϕ and ϕ' in order for ϕ to be included in Φ^*. γ can be defined based on the problem and represents the maximum deviation in performance from the best performing configuration ϕ' that is allowed to enable the exploration of more efficient configurations. In some cases, maximum performance may not be necessary, so energy can be saved by increasing γ. Otherwise, if maximum performance is desired, then γ can be set to 0, so only ϕ' is in Φ^*.

Given that $E(\phi)$ is known, we have the following joint loss function for each ϕ in Φ^*:

$$L_{joint}(\phi, \lambda_E) = (1 - \lambda_E) * L_f(\phi) + \lambda_E * E(\phi) \tag{4.8}$$

where $L(\phi)$ and $E(\phi)$ represent the predicted fusion loss and energy consumption, respectively, of ϕ; and $\lambda_E \in [0.0 - 1.0]$ is the weighting factor that weights the importance of energy consumption vs. performance in the joint optimization. Next, we select ϕ^*, a configuration in Φ^* which lies on the Pareto frontier of the following minimization:

$$\phi^* = \underset{\forall \phi \in \Phi^*}{\arg\min}(L_{joint}(\phi, \lambda_E)) \tag{4.9}$$

After ϕ^* is identified, it is executed to produce the final set of detections $\hat{\mathbf{Y}}$.

4.4 EcoFusion Methodology

We propose EcoFusion, a novel adaptive sensor fusion approach that jointly optimizes performance and energy consumption by identifying the context of an environment before subsequently adapting the model and fusion architecture. Our model can: (i) adapt between using no fusion, early fusion, and late fusion, (ii) select from one or more radar, lidar, or camera sensor inputs, and (iii) execute different types of fusion simultaneously depending on what it determines is the best execution path to minimize loss and energy consumption in the current context jointly.

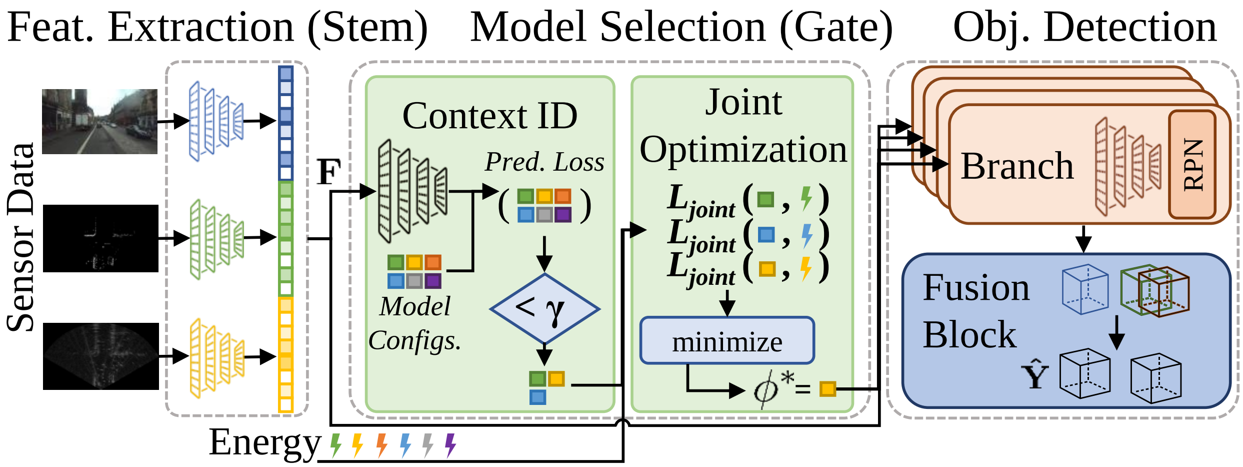


Figure 4.3: Our proposed EcoFusion methodology.

The workflow for EcoFusion is shown in Figure 4.3 and is detailed in Algorithm 2. First, sensor measurements are passed through modality-specific stem models, which produce an initial set of features $\mathbf{F}$ for each sensor. Next, the gate model uses $\mathbf{F}$ and the set of possible model configurations Φ to estimate the loss of each possible configuration for the given inputs. After selecting the candidates for optimization using γ, we pass these candidates Φ^*, their known energy consumption E, and their estimated losses L_f to produce L_{joint} for the optimization function. Then, the ϕ with the lowest L_{joint}, denoted ϕ^*, is selected to execute as is done in Equation 4.9. Since each ϕ represents an ensemble of one or more object detectors, denoted as branches, we run each branch in ϕ^* with its expected inputs and collect the results $\hat{\mathbf{Y}}^*$. These are then fused using our late fusion block, producing a final set of detections $\hat{\mathbf{Y}}$. The following subsections elaborate on the different components in EcoFusion.

4.4.1 Stem Model

The stem models are implemented as a small set of CNN layers that produce an initial set of features for each input modality. The stems are modality-specific, so there is one stem

```
Algorithm 2: EcoFusion Algorithm
  Input: $\mathbf{X}$, $\lambda_E$, $\Phi$, $\gamma$, $E(\Phi)$
  Output: Object Detections $\hat{\mathbf{Y}}$
  Initialize feature vector $\mathbf{F}$ and branch output vector $\hat{\mathbf{Y}}^*$.
  for $s$ in sensors do
      $\mathbf{F}[s] \leftarrow stem(s)$                              // extract features by modality
  $L_f(\Phi) \leftarrow \text{gate}(\mathbf{F}, \Phi)$                          // estimate model losses
  $\Phi^* \leftarrow \rho(L_f(\Phi), \gamma)$                                 // select candidates
  for $\phi$ in $\Phi^*$ do
      $L_{joint}(\phi, \lambda_E) \leftarrow (1 - \lambda_E) * L_f(\phi) + \lambda_E * E(\phi)$
  $\phi^* \leftarrow \arg\min_{\forall \phi \in \Phi^*}(L_{joint}(\phi, \lambda_E))$                      // joint opt.
  for branch in $\phi^*$ do
      $\hat{\mathbf{Y}}^*[branch] \leftarrow branch(\mathbf{F}^*)$                      // pass subset of F
  $\hat{\mathbf{Y}} \leftarrow fusion\_block(\hat{\mathbf{Y}}^*)$                         // fuse branch detections
```

for each type of sensor used. The collection of features F output by the stems is collectively passed to the gate model to identify the context and select the set of branches to execute. Then, F is input to the selected branches.

4.4.2 Context-Aware Gating Model

We implement several gating strategies to estimate the fusion losses of each sensor configuration and facilitate the selection of ϕ^*. The goal of each gating model is to (i) identify the context based on the input features, (ii) estimate the performance of each model configuration in the context, and (iii) compute the optimization result and use it to select ϕ^*. Next, we detail the different methods we implemented for performing steps (i) and (ii). The architectures of our three gating models are shown in Figure 4.4.

Rigid Knowledge-Based Gating Since there exists some domain knowledge as to how each context will affect each sensing modality, we can implement *Knowledge Gating*, where this domain knowledge is used to statically encode the subset of branches to execute for a

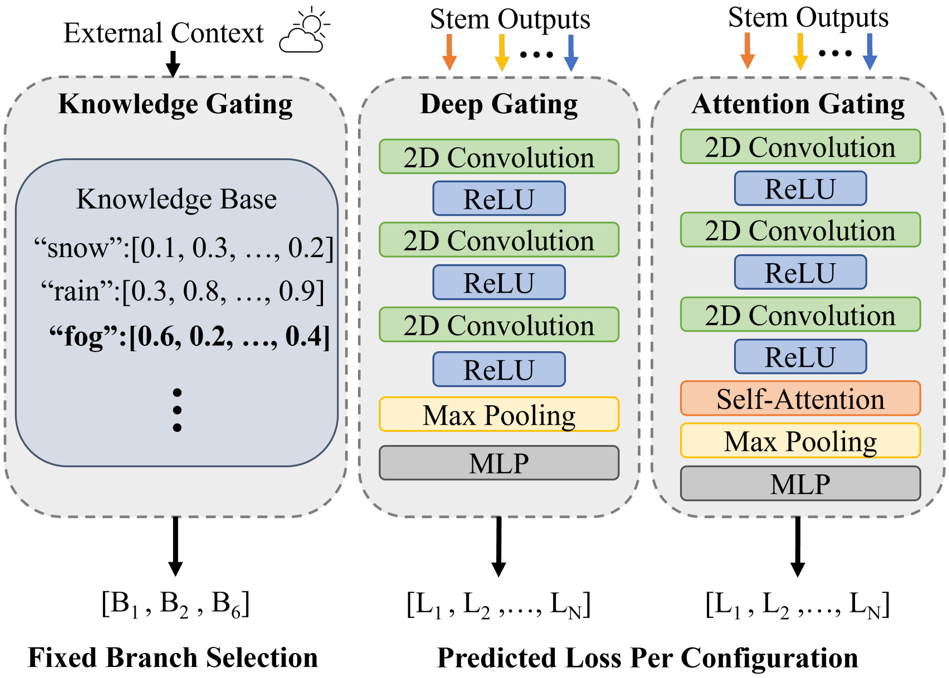


Figure 4.4: Gating Model Architectures.

given context. This assumes the set of possible contexts is finite, and the current context can be identified via external sources. For example, weather information, time of day, and map data can all be used to define the current context. In our approach, we define the set of fixed contexts based on metadata from the RADIATE dataset [1] describing the type of driving data in each sequence. Thus, our set of fixed contexts is: {*city, motorway, junction, rural, snow, fog,* and *night*}. We leverage domain knowledge from the RADIATE paper to rank the relative performance of each sensor in each fixed context. Then, at run-time, the external context information (*e.g.*, data from a navigation/weather system) is used to identify the current context. The top-k ranked branches for that context are selected to be executed and fused. The limitation of this gating strategy is that it requires a *fixed* context definition, potentially limiting performance in cases where contexts are less rigidly defined. With our other gating strategies, we define the context as a *continuous* feature space to enable the modeling of more complex contexts.

Deep Gating In *Deep Gating*, we implement a CNN followed by a multi-layer perceptron (MLP) to model the relationship between the features output from the stems and rank the branches based on their expected performance for this feature set. The outputs of the CNN are flattened to one dimension before being passed to the MLP. Then, the optimization function is run on these outputs. In this gating method, the context can be viewed as a continuous feature space defined by the stem outputs.

Attention Gating In some contexts, certain regions of the feature map may be more informative than others about the scene's context and, consequently, the branch-wise performance. We implement an attention-based gating strategy, denoted as *Attention Gating*, that infers an attention map over the stem features to evaluate this hypothesis. This attention map is used with CNN and MLP layers to model the relationship between branch performance and stem features. We use the visual attention layer proposed in [139] in our implementation.

Loss-Based Gating

We implement this fourth type of gate purely for benchmarking purposes. In this strategy, the *a posteriori* ground-truth loss from each configuration for a given input is used to select ϕ^*. Thus, this implementation is not deployable in the real world but represents the theoretical best-case performance for a gate model that can perfectly predict the fusion loss of every configuration for every input.

4.4.3 Branch Models

The branches in the model take the form of various object detectors. Each branch performs object detection by implementing a Faster R-CNN [138] object detector containing a ResNet-

18 CNN model [140] to extract features from input images and a Region Proposal Network (RPN) to propose object locations across the feature map. The RPN proposals are then fed through a region-of-interest layer that predicts Y^i_{class}, Y^i_{reg} for each box i, as well as the confidence scores for the predicted boxes. We split each ResNet-18 model after the first convolution block, such that the first block becomes the stem, and the remaining three convolution blocks are used in each branch. Each branch can be configured to process either a single sensor or a set of sensors. In this work, we implement one branch for each input sensor and three early fusion branches that fuse both homogeneous and heterogeneous sets of sensors. Using the gate to select the branches, our model can dynamically choose between no fusion, early fusion, late fusion, and combinations of the three.

4.4.4 Fusion Block

The fusion block is implemented via a typical late-fusion algorithm. The detections from any number of branches are first converted to a uniform coordinate system before being statistically processed and fused using the weighted box fusion method from [141]. This process helps refine the accuracy of the bounding box predictions by reinforcing predictions with high confidence and overlap with other predictions.

4.5 Experiments

In our experiments, we used the RADIATE [1] dataset, which provides annotated real-world object detection data from an AV with the following sensors: a Navtech CTS350-X radar, a Velodyne HDL-32e lidar, and a ZED stereo camera. The following classes of objects are annotated in the dataset: {*car, van, truck, bus, motorbike, bicycle, pedestrian, group of pedestrians*}. The dataset consists of various difficult driving contexts (*e.g.*, *rain, fog, snow,* *city, motorway*) that are challenging for typical object detectors. In EcoFusion, we use a 70:30 train-test split across the dataset and train our model with all of the stems and branches enabled using supervised learning. Next, we take the trained stem and branch outputs and use them to separately train the gate model to select the branches that produce the lowest loss for a given stem output ($\mathbf{F}$). We evaluate each model's performance at object detection using average loss and mean average precision (mAP), which is widely used for benchmarking object detection models [138, 142]. We compute the mAP for bounding boxes with an intersection-over-union (IoU) ≥ 0.5, aligning with the PASCAL Visual Object Classes (VOC) Challenge [142]. We calculated the energy consumption of each model configuration $\phi \in \Phi$ on the Nvidia Drive PX2 shown in Figure 4.2. We ignore the energy consumed by the gate models as we measured that they have negligible energy consumption (< 0.005 J) compared to the stems and branches of the model after TensorRT compilation. In all of our experiments, we set $\gamma = 0.5$ as we experimentally determined that it ensures performance at least as good as early and late fusion while enabling energy optimization. However, we note that γ can be tuned based on the requirements for a given application.

4.5.1 Joint Optimization Analysis

We evaluated the trade-off between the performance (model loss) and energy consumption (in Joules) for each gating model in Figure 4.5. We varied λ_E between 0-1.0, where each point in the chart is color-coded according to its λ_E value. As shown, tuning λ_E higher or lower skews the model towards either increasing energy efficiency or increasing performance, respectively, so λ_E should be chosen depending on the requirements for a given application. The configuration for *Loss-Based* that best minimizes both objectives is $\lambda_E = 0.5$ with a loss of 0.966 and energy consumption of 0.844 J. *Attention* and *Deep* have similar Pareto frontiers, but *Attention* achieves better solutions for higher λ_E values while *Deep* achieves slightly lower loss with some low λ_E values. The gap between *Attention/Deep* and *Loss-*

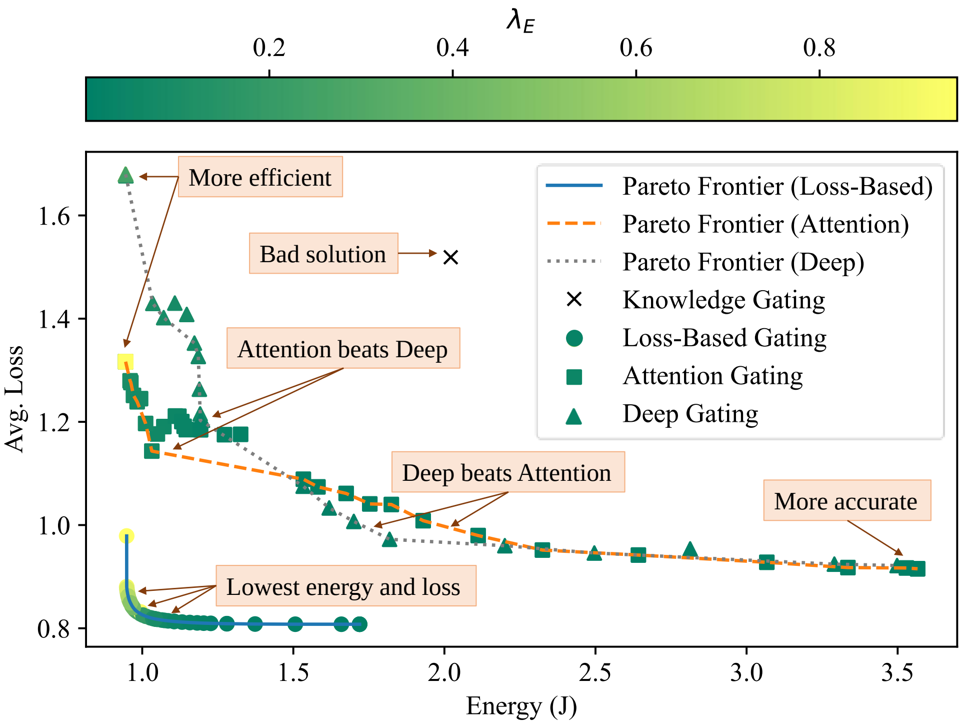


Figure 4.5: Analysis of the energy-loss trade-off of EcoFusion's optimization function with gating models and λ_E values.

Based is likely due to modeling limitations and could potentially be closed using larger or more advanced gate models. For *Attention*, $\lambda_E = 1$ (most energy efficient) results in a loss of 1.317 and an energy consumption of 0.945 J, while $\lambda_E = 0$ (best performing) results in a loss of 0.9153 and an energy consumption of 3.566 J. As shown by the nearly flat trend on the right side of the plot, *Deep* and *Attention* can reduce energy significantly with little effect on loss by tuning λ_E. *Knowledge* is statically programmed such that, for each scenario type, we use domain knowledge to manually select the best sensor combination to use. Due to these constraints, *Knowledge* can be less efficient in some scenarios and is not tunable with our optimization.

4.5.2 Energy and Performance Evaluation

Our results for energy consumption and performance evaluation are shown in Table 4.1. In all of our experiments, early fusion takes in both cameras and lidar as input, while late fusion uses both cameras, lidar, and radar. The energy consumption and latency increase as the fusion method is varied from none to early to late, which is as expected as the latter methods require increasingly larger detection pipelines. The single-sensors are the most efficient, but their mAP scores vary widely from 67% to 79%, likely due to inconsistent performance across scenarios. Early fusion is faster, more efficient, and achieves a higher mAP score and than late fusion; however, early fusion is insufficiently robust in poor driving conditions as will be discussed in Section 4.5.4. EcoFusion with $\lambda_E = 0.01$ achieves higher mAP than all other methods with less energy than late fusion. With $\lambda_E = 0.05$, EcoFusion still outperforms early fusion with less energy usage. As stated in [5], an AV must be able to process inputs at least once every 100 ms (10 frames per second) to ensure safety. In addition to meeting this latency requirement, EcoFusion also executes faster than both early and late fusion, which can improve safety and responsiveness by enabling the AV to process inputs more frequently. With $\lambda_E = 0.01$, EcoFusion achieves a mAP score **5.1%** and **9.5%** higher than early and late fusion, respectively, with **60%** less energy and **58%** lower latency than late fusion.

Fusion Type	Configuration	mAP (%)	Energy (J)	Latency (ms)
None (single sensor)	L. Camera (C_L)	74.48%	0.945	21.57
	R. Camera (C_R)	79.00%	0.945	21.57
	Radar (R)	67.74%	0.954	21.85
	Lidar (L)	70.45%	0.954	21.85
Early Fusion	$C_L + C_R + L$	80.26%	1.379	31.36
Late Fusion	$C_L + C_R + L + R$	77.98%	3.798	84.32
EcoFusion (Ours)	$\lambda_E = 0$	82.92%	3.566	81.49
	$\lambda_E = 0.01$	**84.32%**	**1.533**	**35.14**
	$\lambda_E = 0.05$	**82.16%**	**1.110**	**25.43**

Table 4.1: Energy Consumption and Performance Evaluation

4.5.3 Gating Method Evaluation

Table 4.2 shows mAP, loss, and energy results from evaluating our gating strategies at different λ_E values. With $\lambda_E = 0$, the models tend to pick better-performing branches regardless of their energy consumption. As λ_E increases, the joint optimization significantly reduces energy consumption while keeping loss within γ of the lowest-loss configuration. Although *Knowledge* achieves decent mAP scores, it lacks tunability and thus achieves the same loss and energy consumption for all λ_E; the encoded knowledge would need to be manually updated to adjust the trade-off. *Loss-Based* achieves the lowest loss and energy consumption but a lower mAP than *Deep* and *Attention*. This result is likely because loss is not perfectly correlated with mAP score; mAP primarily scores object classification over properly aligned bounding boxes, while loss is measured across both classification and box regression. Overall, *Attention* performs slightly better than *Deep* and offers the best trade-off of performance and energy.

λ_E	Gating Method	mAP (%)	Avg. Loss	Energy (J)
0	*Knowledge*	82.43%	1.519	**2.021**
0	*Deep*	82.68%	**0.915**	3.556
0	*Attention*	**82.92%**	**0.915**	3.566
0	*Loss-Based*	82.50%	0.808	1.719
0.01	*Knowledge*	82.43%	1.519	2.021
0.01	*Deep*	83.72%	1.124	**1.457**
0.01	*Attention*	**84.32%**	**1.089**	1.533
0.01	*Loss-Based*	81.65%	0.809	1.280
0.1	*Knowledge*	**82.43%**	1.519	2.021
0.1	*Deep*	81.98%	1.432	1.008
0.1	*Attention*	79.72%	**1.280**	**0.960**
0.1	*Loss-Based*	79.70%	0.818	1.044

Table 4.2: Gating method evaluation.

4.5.4 Scenario-Specific Evaluation

Figure 4.6 shows loss and energy results for different driving scenarios in the dataset. We evaluated no fusion (radar-only), early fusion, late fusion, and EcoFusion with *Attention Gating*. As shown in the figure, EcoFusion performs similarly to late fusion in terms of loss across all scenarios. It is also clear that early fusion performs poorly in the difficult driving conditions present in the *Fog* and *Snow* scenarios. Late fusion is more robust and achieves relatively good performance across scenes; however, late fusion also consumes significantly more energy than all other methods. In contrast, EcoFusion's energy efficiency is on-par with early fusion and is significantly lower than that of late fusion. No fusion was the most energy-efficient but also had the highest overall loss.

4.6 Discussion

4.6.1 Practicality

Since we evaluated EcoFusion with the industry-standard Nvidia Drive PX2 autonomous driving platform, it is clear that EcoFusion can save energy on real-world AV hardware while meeting real-time latency constraints. Furthermore, by achieving better object detection performance with lower latency, EcoFusion improves safety and robustness over existing methods. Our evaluation on a diverse driving dataset proves that EcoFusion is robust across scenarios and is thus more practical for real-world driving. To implement EcoFusion on a real driving system, the designer would first need to train the model on the appropriate dataset before selecting the best λ_E and γ for their design requirements. Then, the model can be compiled for hardware using TensorRT or a similar library and integrated into the AV stack.

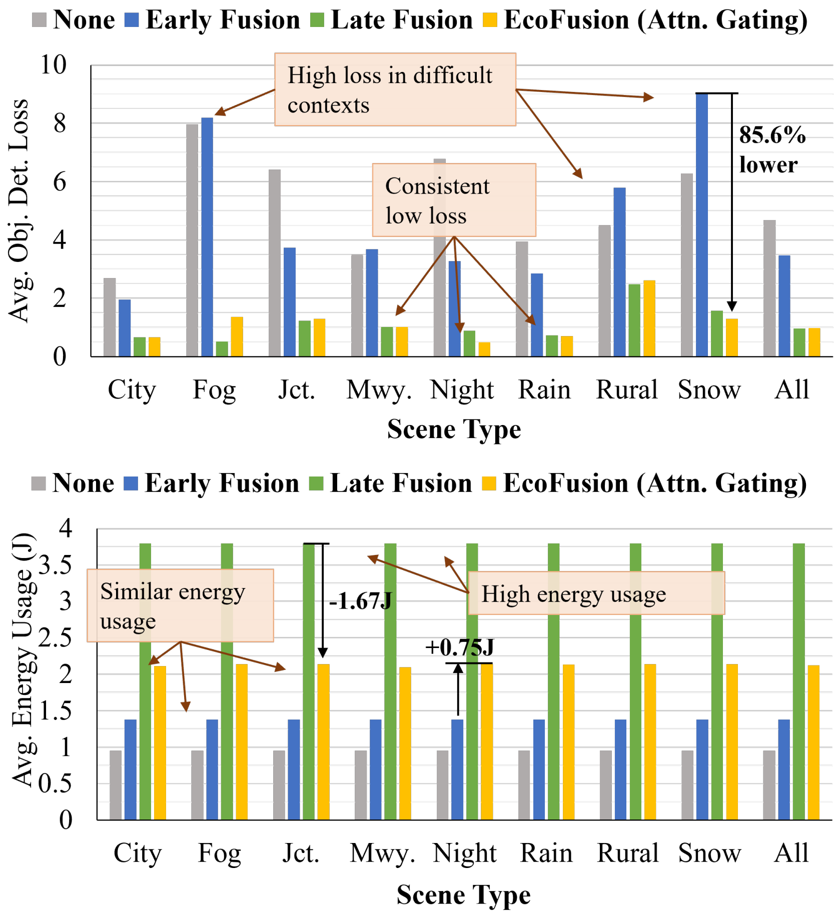


Figure 4.6: Average loss and energy consumption per scenario for each fusion method. Junction and Motorway are abbreviated as Jct. and Mwy., respectively. EcoFusion achieves low loss across scenes with 43.7% lower energy consumption than late fusion.

4.6.2 Sensor Clock Gating

More energy could be saved by disabling unused sensors using clock gating. The Navtech CTS350-X radar uses 24 W [18], the Velodyne HDL-32E lidar uses 12 W [19] and the ZED camera uses 1.9 W [143], so reducing sensor energy usage can significantly improve AV efficiency. Temporal modeling can enable the context to be estimated across time instead of for a single input, allowing clock gating for specific periods. In Table 4.3, we analyze the benefits of sensor clock gating with our *Knowledge Gating* approach in each driving scenario since it uses external context to inform sensor selection. We also show baseline results with late

Fusion	Avg. Energy Consumption (J) by Scene Type								
Method	*City*	*Fog*	*Jct.*	*Mwy.*	*Night*	*Rain*	*Rural*	*Snow*	*Overall*
Late Fusion	13.27	13.27	13.27	13.27	13.27	13.27	13.27	13.27	13.27
EcoFusion (Ours)	**5.45**	13.96	**2.87**	**2.87**	**12.10**	13.29	**3.81**	13.96	**6.45**
EcoFusion Savings	**58.91%**	-5.15%	**78.40%**	**78.40%**	**8.81%**	-0.09%	**71.28%**	-5.15%	**51.41%**

Table 4.3: Combined sensor and AV hardware platform energy consumption in each driving scenario.

fusion across the four sensors. Using the power consumption P and measurement frequency f of each sensor s, we estimate the energy that could be saved by stopping measurements without slowing the motor's rotation. We cannot completely power gate the rotating lidar and radar sensors because they have inertia and require several seconds to get back up to speed from a stand-still, which can compromise safety. We model the energy consumption E_s of each sensor and the total energy consumption E_{total} as follows:

$$E_s = (P_s^{meas.} + P_s^{motor}) * 1/f_s, \quad P_s^{meas.} = P_s - P_s^{motor} \tag{4.10}$$

$$E_{total} = E(\phi) + \sum_{s \in \phi} E_s \tag{4.11}$$

where ϕ is the model configuration defined for the context. After our calculation, we set $P^{meas.} = 0$ to simulate clock gating of the sensor. The Navtech CTS350-X consumes 2.4 W to spin the motor, so its $P^{meas.} = 21.6$ W. Based on comparable lidar motor models, we estimate the Velodyne HDL-32E's $P^{meas.} = 9.6$ W. As shown in Table 4.3, EcoFusion would use up to **78.40%** less energy than late fusion in common driving scenarios. EcoFusion uses slightly more energy than late fusion in more difficult driving scenarios, but these scenarios are rare, so overall energy consumption is still lower. On average, clock gating unused sensors with EcoFusion uses **51.41%** less energy than running all sensors with late fusion and **43.90%** less energy than EcoFusion without sensor clock gating. Since *Knowledge Gating* is less effective than *Deep* and *Attention*, Chapter 5 explores methods for enabling smart sensor clock gating with these gate models via intermittent sensing and context re-identification.

4.7 Summary

This chapter introduces EcoFusion — a novel adaptive sensor fusion approach that uses contextual information to adapt its architecture and jointly optimize performance and energy consumption. We show that EcoFusion outperforms early and late fusion in terms of mAP (**84.32%** vs. 80.26% and 77.98%), with similar energy consumption and latency to early fusion. We also demonstrate that in difficult driving contexts, EcoFusion is more robust than early fusion (up to **85.6%** lower loss) and more efficient than late fusion (**60%** less energy). We additionally propose and evaluate multiple gating strategies and find that a learned strategy outperforms a knowledge-based strategy. Overall, we show that an energy-aware adaptive sensor fusion approach can significantly improve the energy efficiency and perception performance of AVs. The next chapter explores how this methodology can be extended to a broader system-wide optimization via intermittent sensing and a reconfigurable hardware platform.

Chapter 5

Hardware/Software Reconfiguration for Energy-Efficient Sensor Fusion

5.1 Introduction

As discussed in Chapter 4, advanced deep-learning models and multiple heterogeneous sensors (e.g., cameras, radars, and lidars) are necessary for perception across different weather and lighting conditions. The previous chapter demonstrated how context-aware dynamic sensor fusion architectures (EcoFusion) can effectively manage the trade-off between perception performance and energy-efficiency in perception algorithms, allowing joint optimization of both objectives and thus robust and energy-efficient performance across driving scenarios.

5.1.1 Research Challenges

Still, EcoFusion focuses on algorithmic optimizations, while large energy consumers, such as the sensors and the GPU-based hardware computation platform, are not included in the energy optimization. If the complete perception pipeline (sensors, hardware platform, and algorithms) can be included in the joint optimization, greater energy savings could be achieved. In summary, the key challenges targeted by this chapter include: (i) perceiving effectively in complex and adverse driving scenarios; (ii) reducing the energy consumption of the complete perception system, including sensors, hardware, and algorithms; and (iii) adapting the system configuration to different contexts, enabling energy efficiency without compromising performance.

5.1.2 Novel Contributions

To overcome these challenges, we propose using field-programmable gate arrays (FPGAs) with Deep-learning Processing Units (DPUs) [144] for energy-efficient, low-latency runtime model reconfiguration in hardware. FPGAs provide lower latency, lower power consumption than GPU-based hardware, and greater flexibility than application-specific integrated circuits (ASICs). DPUs are soft-logic compute cores programmed onto FPGAs, enabling them to achieve negligible model switching latency and flexible accelerator configuration compared to typical reprogramming methods. By leveraging the flexibility of FPGAs with the high-throughput and low context-switching latency of DPUs, we can achieve energy-efficient real-time perception via end-to-end, system-wide optimizations. Thus, we propose CARMA: a context-aware dynamic sensor fusion approach that uses *runtime model reconfiguration* to adapt its DPU-based architecture on an FPGA. CARMA implements a tunable energy-performance optimization over the complete perception system, including the sensors, model architecture, and computation hardware platform, to maximize energy savings without compromising perception performance. To our knowledge, this is the first work to propose energy-efficient sensor fusion via context-aware runtime model reconfiguration on FPGAs. Our major contributions can be summarized as follows:

1. We propose CARMA, an approach for dynamically reconfiguring a complete perception and sensor fusion system for object detection at runtime using contextual information from the sensors. CARMA uses DPUs on FPGA to enable runtime model reconfiguration with negligible model switching latency.

2. We propose a method for intermittently performing state estimation and context identification to enable intelligent sensor and submodel clock gating to maximize energy efficiency.

3. We use a tunable joint optimization between perception performance and system energy consumption to maximize energy efficiency while minimizing perception performance impacts.

4. We show that CARMA significantly reduces system-wide energy usage vs. state-of-the-art sensor fusion methods and achieves equivalent or better object detection performance across diverse autonomous driving scenarios with up to **1.3**× inference speedup and 73% lower energy consumption.

5.2 Related Works

5.2.1 Adaptive Computing Systems on FPGA

Self-adaptive systems can modify their runtime behavior according to changing environments and system goals. [145] presents a dynamically reconfigurable convolutional neural network (CNN) accelerator optimized for throughput. In [146], an FPGA reconfigures at runtime to use a lower power design when the battery level decreases. However, it has limitations such as latency overhead proportional to the size of the bitstream file, which restricts reconfiguration to small components, and the time and knowledge needed to create hardware designs for each reconfiguration option.

The Xilinx DPU with the Vitis AI software stack enables adaptable and efficient AI inference for FPGAs. Users can reconfigure the NN model at runtime with minimal latency overhead by changing the input of the DPU without altering the FPGA's hardware logic. [147] explored a DPU-based energy-efficient hardware accelerator. However, it failed to optimize energy efficiency system-wide and its approach was too simplified to handle complex environments.

5.2.2 Energy-Performance Optimization

Several works have explored methods for managing the trade-off between energy consumption and the performance of deep learning algorithms at runtime, e.g., dynamic width neural network [135] and dynamic model selection for classification [148]. However, these works only support a single input modality and are restricted to image classification. More recent works address this limitation by applying optimizations to multi-sensor fusion for perception [149, 130]. The methodology proposed in Chapter 4 proposes a dynamic-width sensor fusion model that aims to select lower energy submodels while maintaining performance. Although this approach incorporates multimodality, it only optimizes the object detection model parameters and omits *system-wide* energy optimizations. This chapter proposes extending EcoFusion to a heterogeneous FPGA-driven compute platform to maximize the energy saved by dynamic model selection while applying system-wide energy optimizations to reduce energy usage further.

5.2.3 Intermittent Sensing and Control in Autonomous Systems

Due to the energy constraints of many AS, several methods for intermittent sensing and control have been proposed to reduce energy consumption without compromising performance [129, 86]. For example, [129] proposes an adaptive sensor fusion algorithm for indoor robot localization that adjusts the sensing rate of different sensors to match the dynamics of the physical environment while saving energy. In the control domain, [86] uses safety guarantees to determine if actuation can be skipped for brief intervals to save energy without entering an unsafe state, reducing energy usage. Similarly, [150] proposes using an intermittent control strategy for autonomous driving to emulate human-like control behavior. It improves stability, robustness, and energy efficiency over continuous control approaches by only applying control outputs when errors exceed set thresholds. Like these works, CARMA targets energy efficiency by intermittently reconfiguring the model architecture and the set of active sensors to match the current environment context.

5.3 Methodology

5.3.1 Problem Formulation

Object Detection Model

CARMA uses the same object detection problem formulation as EcoFusion. Please refer to Chapter 4.3 for more details. To summarize, the goal of the object detector ϕ is to use the sensor measurements $\mathbf{X}$ to accurately identify the objects $\mathbf{Y}$ in the environment:

$$\mathbf{Y} = \phi(\mathbf{X}), \text{ where } \mathbf{Y} = \{\mathbf{Y}^i_{class}, \mathbf{Y}^i_{reg}\}_{i=1\ldots d} \tag{5.1}$$

where $\mathbf{Y}^i_{class}, \mathbf{Y}^i_{reg}$ denote the class and bounding box, respectively, of object i. Our proposed approach uses context to identify the best combination of early and late fusion to improve the accuracy of the resultant predictions across driving contexts. As such, the object detection model becomes:

$$\hat{\mathbf{Y}} = \phi(\phi_1(\mathbf{X}_1),\, \phi_2(\mathbf{X}_2),\, \ldots,\, \phi_3(\psi(\mathbf{X}_2,\, \mathbf{X}_s))) \tag{5.2}$$

Where ϕ_1 and ϕ_2 represent single-sensor object detectors, ϕ_3 is a multi-sensor object detector using early fusion, and ϕ is the late fusion function for fusing the detectors' outputs to obtain $\hat{Y}$. Section 5.3.2 describes how CARMA identifies context and selects the appropriate model configuration.

Energy Model

CARMA's energy model aligns with that of EcoFusion with extensions to model sensor clock gating and system-wide energy usage. We model the energy usage of the complete AV driving system E_{sys} as the total energy consumed by the sensors E_s and the execution of the algorithm E_a on the hardware platform.

$$E_{sys} = E_s + E_a \tag{5.3}$$

We omit factors such as drivetrain energy usage and battery lifetime as these factors have been studied in existing work [27, 25, 151, 30] and can be used in conjunction with our approach. Typical AS contain some combination of static sensors (*e.g.*, cameras, ultrasonic sensors, front-facing radar) and rotating sensors (*e.g.*, spinning top-mounted lidar). The energy consumption per sensor $s \in S$ can be computed from the measurement power $P^{meas.}_s$, measurement frequency f_s, and, for spinning sensors, the motor power P_s^{motor}, as follows:

$$E_s = (P_s^{meas.} + P_s^{motor}) * 1/f_s \tag{5.4}$$

To reduce the energy consumption of the complete system, we clock gate sensors unused in the current visual context. The lidar and radar sensors in our testbed, discussed in Section 5.4.1, are top-mounted spinning sensors, while the cameras are fixed sensors without motors. Since the lidar and radar have inertia and require several seconds to start and stop rotating, we assume that we only clock gate the measurement components while keeping the motor spinning so they can be quickly re-enabled to ensure safety. As discussed in Chapter 4.6.2, the Navtech CTS350-X radar uses 24 W, while the Velodyne HDL-32E lidar uses 12 W and the ZED camera uses 1.9 W. The Navtech CTS350-X needs 2.4 W to spin the motor, so $P_{radar}^{meas.} = 21.6$ W. Using comparable lidar motor models for the Velodyne HDL-32E, we estimate $P_{lidar}^{meas.} = 9.6$ W.

Since our object detection model is reconfigurable, the algorithm energy consumption E_a can be computed as:

$$E_a(\phi, \mathbf{X}) = P_a(\phi, \mathbf{X}) * t(\phi, \mathbf{X}), \tag{5.5}$$

where $t(\phi, \mathbf{X})$ represents the processing latency in seconds and $P_a(\phi, \mathbf{X})$ represents the power consumption in Watts of processing input $\mathbf{X}$ through the current model configuration ϕ on the hardware platform. We measured the power and latency of each model configuration on our hardware platform, the Xilinx Kria KV260 FPGA, to compute E_a offline for use in our multi-objective optimization.

Multi-Objective Optimization

CARMA implements a tunable joint optimization between system-wide energy consumption and model performance to enable it to minimize energy without compromising performance. Similar to EcoFusion, CARMA uses a λ_E term to allow model designers to specify the preference for energy efficiency over performance depending on the application of the system. Given that we know the expected performance L of configuration ϕ for an input $\mathbf{X}$, denoted as $L(\phi, \mathbf{X})$, and the expected system-wide energy consumption of that configuration $E_{sys}(\phi, \mathbf{X})$, our optimization can be formulated as:

$$L_{opt}(\phi, \mathbf{X}) = L(\phi, \mathbf{X}) * (1 - \lambda_E) + E_{sys}(\phi, \mathbf{X}) * \lambda_E \tag{5.6}$$

$$\phi^*(\mathbf{X}) = \underset{\phi \in \Phi}{\arg\min}(L_{opt}(\phi, \mathbf{X})), \tag{5.7}$$

where $\phi^*(\mathbf{X})$ represents the model configuration that best minimizes the joint optimization loss L_{opt} for input $\mathbf{X}$ for the given λ_E. [43] used a similar optimization to select which branches to execute, with all other system components remaining fixed. However, our proposed approach includes clock gating of unused sensors and stems, drastically increasing the potential energy savings and enabling system-wide optimization.

5.3.2 System Architecture

CARMA's architecture is shown in Fig. 5.1. CARMA consists of a runtime reconfigurable multi-branch sensor fusion model for object detection. Section 5.3.4 elaborates on our runtime reconfiguration approach on hardware, while the following text describes our sensor fusion model. Similar to EcoFusion, the model consists of four key components, (i) feature extraction, (ii) context identification, (iii) submodel selection, and (iv) output fusion. First, multi-modal sensor data is processed by modality-specific *Stem* models to extract an initial

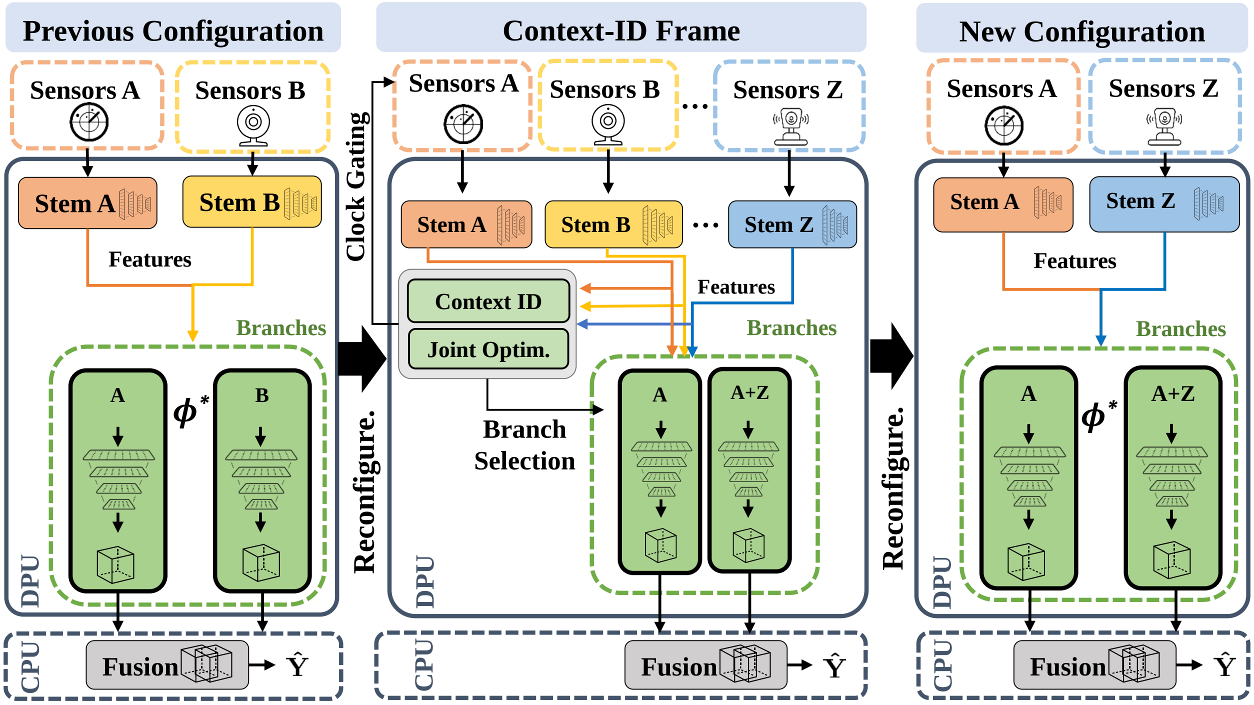


Figure 5.1: CARMA System Architecture and Reconfiguration Workflow

set of features for each sensor. These features are then used by the *Gate* model to identify the current visual context. This context is used to select the set of submodels (*Branches*) to execute that optimizes performance and energy efficiency. Each active branch outputs a set of object detections collected and fused by the *Fusion Block* to produce a final set of refined detections. We detail each of these model components below.

Stem and Branches

We use the popular Single Shot MultiBox Detector (SSD) [152] for object detection. SSD processes sensor data using a CNN and outputs a bounding box (location) and classification for each object it detects. It outperforms Faster R-CNN [138] in terms of both speed and performance on well-known benchmarks. SSD is similar to Faster R-CNN in architecture but is more hardware-friendly because region proposal and object detection are performed in a single pass by the CNN, while Faster R-CNN is a two-stage model that requires a separate

Region Proposal Network (RPN). Additionally, SSD has a smaller model size and fewer intermediate feature maps to store and transfer between different layers of the network. These factors make SSD faster to execute on FPGAs since it requires fewer hardware resources and lower memory bandwidth.

In our proposed architecture, the *stem* models serve as modality-specific preprocessors, so we use the first six layers of SSD's ResNet-18 backbone to build each stem. Each stem produces a set of features **F** for one sensor. The remaining 23 layers of SSD form each branch. Thus, when data from a sensor passes through one stem and one branch, it equates to processing the sensor through a complete SSD model. We implement single-sensor branches for four sensor inputs: two front-facing cameras, one top-mounted lidar, and one top-mounted radar. We also implement three early-fusion branches that take multiple sensors as input: dual camera, lidar and radar, and dual camera with lidar. These branches include a single merge convolution layer to combine the sensors across the channel dimension before continuing with processing.

Context Identification and Gating

To identify the current visual context and perform branch selection, we use the three *gate* models proposed in Chapter 4. To summarize, the *knowledge* gate uses fixed domain-knowledge rules to select submodels using external contextual information (e.g., weather, time of day, road type). The rules encode domain knowledge on the sensor modalities least likely to be degraded by current environmental factors such as rain, snow, fog, or lighting. Meanwhile, the *deep* gate uses a 3-layer CNN to infer the current context from the stem output features and directly output the set of branches it infers will perform best in the current visual context. Here, context refers to an abstract visual state estimate generated within the CNN's hidden layers, while the gate output indicates which branches to execute. The *attention* gate is the same as the *deep* gate with the addition of a self-attention layer.

Given the set of all possible model configurations Φ, the objective of the gate is to estimate the performance L of each configuration ϕ for the current set of input features F:

$$L(\Phi, \mathbf{F}) = \pi(\phi, \mathbf{F}), \forall \phi \in \Phi \tag{5.8}$$

$$\rho(L(\Phi, \mathbf{F}), \gamma) = \{\phi \in \Phi \text{ s.t. } L(\phi, \mathbf{F}) \leq L(\phi', \mathbf{F}) + \gamma\} \tag{5.9}$$

$$\Phi^* = \rho(L(\Phi, \mathbf{F}), \gamma), \tag{5.10}$$

where π represents the gating model and ρ represents a function for identifying the set Φ^* of top performing configurations with an estimated error within γ of the best performing configuration ϕ'.

Fusion Block

The objective of the fusion block is to fuse the object detections from the active branches to produce a more accurate set of final bounding box predictions. In this chapter, we use *weighted boxes fusion* [141], which uses the confidence scores of all proposed boxes to average the boxes into a refined set of object detections. The fusion block in CARMA is executed on the CPU, as shown in Fig. 5.1, because it involves complex program logic, which is better supported on the CPU compared to DPU. The CPU can also create a thread for box fusion, which utilizes idle CPU resources during DPU inference.

5.3.3 Hardware Design Choices

Although our proposed sensor fusion methodology can be used across different hardware platforms, the characteristics of the computing platforms must be carefully considered to balance high performance and energy efficiency. FPGAs are superior to GPU and ASIC platforms for

AS due to increased energy efficiency and flexibility. They process large amounts of data with minimal power and can be reprogrammed to accommodate changes. However, safety-critical real-time environments necessitate systems with high throughput and low context-switching latency to adapt to changing environments. We discuss the key benefits of our hardware design choices below.

High Throughput

In AS, data must be processed in real-time with low latency to ensure safe and efficient vehicle operation. Typically, AS must process data at a minimum rate of 10 frames per second (fps) [5] to enable accurate control in changing environments. This requires high parallelism of data processing with constrained resources. The DPU features user-configurable parameters to optimize resource utilization and to select which features are needed for a given deployment scenario. In the DPU, we can configure three dimensions of parallelism for convolutions: pixel, input channel, and output channel parallelism.

Fast Context Switch Interval

CARMA changes branch configurations (context switches) at runtime. Fast context switch intervals are necessary to handle various tasks and events that may occur during vehicle operation. CARMA uses Vitis AI Runtime to load the instruction files into the DPU for inference and switch the context by changing the calling threads corresponding to different configurations. Loading of model instruction files and inference are performed simultaneously, reducing the context switch time to the time of the thread switch (less than 1ms), while traditional FPGA runtime reconfiguration waits until the new bitstream is fully deployed on-board. Since each model file is <25MB and our system has 4GB on-board memory, our system can store all model configurations in DDR memory.

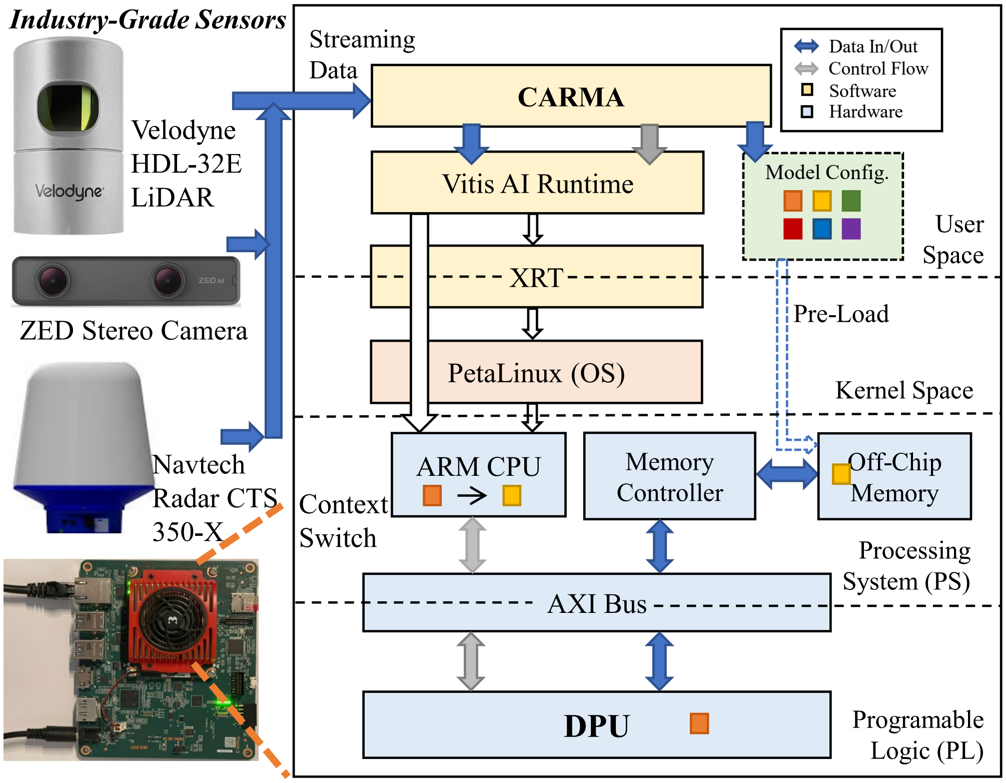


Figure 5.2: CARMA System Stack and Experimental Testbed

5.3.4 Hardware Execution Model

Fig. 5.2 illustrates our hardware execution model. CARMA runs in the application layer on PetaLinux and controls our complete sensor fusion system. It uses the Vitis AI Runtime, a set of high-level APIs, to interact with the DPU. Xilinx Runtime (XRT) is a set of low-level APIs that connect the User Space and Kernel Space and control the hardware. The on-chip processor (Arm CPU) serves as the hardware host control node and controls the DPU, services interrupts, and coordinates data transfers. The processing system (PS) connects to the DPU via the Advanced eXtensible Interface (AXI) bus for transferring data and control signals. When initializing the system, the compiled models for all sensor-fusion configurations are loaded into the off-chip memory, waiting to be called. At runtime, the DPU fetches compiled instructions from off-chip memory to control the operation of the computing engine on the DPU.

5.3.5 Runtime Workflow and Intermittent Context Identification

Several works have demonstrated safe and effective intermittent perception and control approaches, as discussed in Section 5.2.3. These approaches are intuitive since real-world visual contexts often remain the same for several seconds, especially in the case of broad visual contexts like *rainy weather* or *night driving*. We propose using *intermittent* context-identification to enable broader energy optimizations such as clock-gating unused sensors and stem models for brief periods before re-enabling them to identify the current context. CARMA can directly integrate with existing methods for safe intermittent perception since they use similar strategies, such as clock gating, to control sensing frequency.

To reduce the overhead of context identification and switching, we propose the Context-ID Frame design, shown in Fig. 5.1. In sensor fusion mode, we only execute the stems and branches needed for a particular model configuration, minimizing energy consumption. In Context-ID mode, we reconfigure the DPU to the Context-ID Frame to select the next model configuration. The following two algorithms describe the workflow of our proposed approach. Alg. 3 shows the typical operation of CARMA. For each time step t, data is retrieved from the active set of sensors and processed by the current branch configuration ϕ^* to produce the output detections $\hat{\mathbf{Y}}$. T_c represents the context re-identification interval; when $t/T_c = 0$, execution transfers to Alg. 4 for the next time step $t + 1$. Here, T_c can be dynamically configured by an intermittent algorithm, such as those from Section 5.2.3. In Alg. 4, all sensors and stems are activated, and the sensor features F are passed to the gate module π to estimate the loss of each branch configuration. The lowest loss branches are selected by ρ as described in Equation 5.10. Then, this set Φ^* is passed to the joint optimization to identify the optimal configuration ϕ^*. The outputs of the active branches are fused to produce $\hat{\mathbf{Y}}$. After this step, we clock gate the unused sensors, switch to the new model configuration containing just ϕ^*, and continue executing Alg. 3 with the new ϕ^* and *active_sensors* at the next time step.

Algorithm 3: Runtime Sensor Fusion Algorithm

```
Input: $t$, $\phi^*$, $active\_sensors$, $T_c$
Output: Object Detections ($\hat{\mathbf{Y}}$)
Initialize feature vector $\mathbf{F}$ and branch output vector $\hat{\mathbf{Y}}^*$
for $s$ in $active\_sensors$ do
    $X_s \leftarrow s(t)$                                          // data input
    $\mathbf{F}[s] \leftarrow stem_s(X_s)$                          // extract features
for $branch$ in $\phi^*$ do
    $\hat{\mathbf{Y}}^*[branch] \leftarrow branch(\mathbf{F}^*)$     // pass subset of F
$\hat{\mathbf{Y}} \leftarrow fusion\_block(\hat{\mathbf{Y}}^*)$       // fuse detections
if $t/T_c = 0$ then
    $\phi^*, active\_sensors \leftarrow$ Algorithm2$(t+1)$
```

Algorithm 4: Context ID and Reconfiguration Algorithm

```
Input: $t$, $\lambda_E$, $\Phi$, $\gamma$, $E_{sys}(\Phi)$, $all\_sensors$
Output: Object Detections ($\hat{\mathbf{Y}}$), $\phi^*, active\_sensors$
Initialize feature vec. $\mathbf{F}$ and output vec. $\hat{\mathbf{Y}}^*$
for $s$ in $all\_sensors$ do
    $X_s \leftarrow s(t)$                                          // data input
    $\mathbf{F}[s] \leftarrow stem_s(X_s)$                          // extract features
$L(\Phi) \leftarrow \pi(\mathbf{F}, \Phi)$                          // estimate model losses
$\Phi^* \leftarrow \rho(L(\Phi), \gamma)$                           // select candidates
for $\phi$ in $\Phi^*$ do
    $L_{joint}(\phi) \leftarrow (1 - \lambda_E) * L(\phi) + \lambda_E * E_{sys}(\phi)$
$\phi^* \leftarrow \arg\min_{\forall \phi \in \Phi^*}(L_{joint}(\phi))$   // joint opt.
$load\_branches(\phi^*)$                                          // reconfiguration
for $branch$ in $\phi^*$ do
    $\hat{\mathbf{Y}}^*[branch] \leftarrow branch(\mathbf{F}^*)$     // pass subset of F
$\hat{\mathbf{Y}} \leftarrow fusion\_block(\hat{\mathbf{Y}}^*)$       // fuse detections
Initialize empty set $active\_sensors$
for $s$ in $all\_sensors$ do
    if $\phi^*$ requires $s$ then
        $active\_sensors \leftarrow active\_sensors \cup \{s\}$
    else
        $clock\_gate(s)$                                           // clock gate sensors
        $disable\_stem(stem_s)$                                    // reconfiguration
```

5.4 Experiments

5.4.1 Experimental Setup

CARMA can be applied to any multi-sensor AS to enable energy-efficient perception. In our experiments, we evaluate CARMA on a popular AS use case: autonomous driving for AVs. Our hardware testbed is shown on the left side of Fig. 5.2. We use the Xilinx Kria KV260 FPGA as our computing platform. Due to its portability and compatibility, our design could feasibly be implemented on Xilinx automotive-grade FPGAs in a similar manner. Each model is trained on the RADIATE dataset [1], which contains three hours of high-resolution radar, lidar, and stereo camera data across challenging perception contexts. We compare against Faster R-CNN object detectors for single sensor inputs, early and late multi-sensor fusion, and EcoFusion [43]. To measure the object detection performance of each model, we use the object detection loss function from [153], which combines bounding box loss with classification loss. The object detection metrics we present are for a Faster R-CNN variant of our model trained using the same hyperparameters as [43] for fairer comparison with EcoFusion. However, we verified experimentally that the SSD-based model achieves 50% lower average loss and consumes 15% less energy than the Faster R-CNN version. We used built-in functions in the host code and system commands to measure the end-to-end latency and power consumption of different configurations.

5.4.2 Performance on FPGA

We compare the object detection performance and energy consumption of different fusion techniques in Table 5.1. Across different gating and λ_E configurations, CARMA achieves lower average energy usage and loss than almost every early fusion, late fusion, and single sensor model. The only exceptions were the camera-only configurations, which had higher losses than our method but lower energy usage due to the efficiency of the camera sensors. Notably, with an equivalent model loss, CARMA ($\lambda_E = 0, deep$) achieves a **41.3%** reduction in energy compared to EcoFusion ($\lambda_E = 0, attn$). With a higher $\lambda_E = 0.01$ for both models, CARMA achieves **73.7%** lower energy usage with only a 3.2% higher loss than EcoFusion. EcoFusion's inability to account for sensor energy or apply sensor and model clock gating leads to higher average energy consumption, putting it on par with high-energy early fusion and late fusion variants. Regarding latency, CARMA achieves **6%-33%** speed-up compared to EcoFusion, with higher λ_E values resulting in lower model latencies. The results also illustrate the trade-offs for each sensing modality; branches using radar use significantly more energy than camera-only or lidar branches, since radar is less energy-efficient. The lidar and radar branches have higher average loss and energy consumption than the camera branches; however, these sensor modalities are more reliable in contexts where cameras can fail. This point is supported by the lower loss achieved by the late fusion model.

Fusion Type	**Configuration**	**Avg. Loss**	**Energy (J)**	**Latency (ms)**
None	Radar (R)	2.858	6.73	14.2
	Lidar (L)	4.682	3.73	14.2
	Camera (C)	1.680	1.81	14.2
Early	$R + L$	2.784	9.16	17.1
	$C_L + C_R$	1.203	2.31	17.1
	$L + C_L + C_R$	3.476	3.73	19.7
Late	$R + L + C_L + C_R$	0.967	10.48	42.6
EcoFusion [43]	$\lambda_E = 0, attn$	0.915	10.41	54.0
	$\lambda_E = 0.01, attn$	0.924	10.36	48.0
	$\lambda_E = 0.1, attn$	1.147	10.18	27.7
CARMA (Ours)	$\lambda_E = 0, attn$	0.915	7.35	51.9
	$\lambda_E = 0, deep$	0.915	6.12	51.2
	$\lambda_E = 0.0001, attn$	0.920	6.68	50.2
	$\lambda_E = 0.001, deep$	0.944	3.31	42.6
	$\lambda_E = 0.001, attn$	**0.959**	**3.23**	**38.5**
	$\lambda_E = 0.01, deep$	**0.954**	**2.73**	**36.1**

Table 5.1: Performance and energy comparison between different fusion methods. Results for CARMA are with a T_c of 30 samples.

5.4.3 Impact of Different Context Identification Intervals

Table 5.2 shows how the performance characteristics of CARMA would vary at various context identification intervals under an intermittent method such as those in Section 5.2.3. As mentioned in Section 5.3.5, T_c determines the duration to run Alg. 3 before switching to Alg. 4 to re-identify the context and reconfigure the model. Alg. 4 requires enabling all sensors and stems, increasing energy usage every T_c samples. Table 5.2 shows the impacts on energy usage and model responsiveness for different values of T_c under the assumption that sensor data is received at 30 fps.

Context ID Interval (T_c samples)	1	3	10	30	100
Interval @ 30 fps (s)	0.03	0.1	0.33	1	3.33
Average Energy (J)	10.39	5.1	3.25	2.73	2.54

Table 5.2: Energy savings for different context ID frequencies for CARMA ($\lambda_E = 0.01, \mathit{deep}$) where the Context ID block is executed every T_c samples.

Evaluating the context every time-step eliminates clock gating, resulting in the highest energy usage but the fastest model switching interval. With $T_c = 3$, clock gating reduces average energy by over 50%. Values of $T_c > 3$ provide more energy savings with diminishing returns and reduce the model's ability to respond quickly to scenario changes. In our experimental results, we assume the intermittent sensing algorithm sets $T_c = 30$, thus reconfiguring our model every 1.0s. From our analysis, CARMA tends to change configurations less often with a higher λ_E; for example, CARMA with ($\lambda_E = 0.01, \mathit{deep}$) and ($\lambda_E = 0.001, \mathit{deep}$) only used 5 and 11 different configurations across all driving contexts, respectively. This finding illustrates how contexts can remain consistent for long periods, so intermittent context re-identification can increase energy savings without impacting performance.

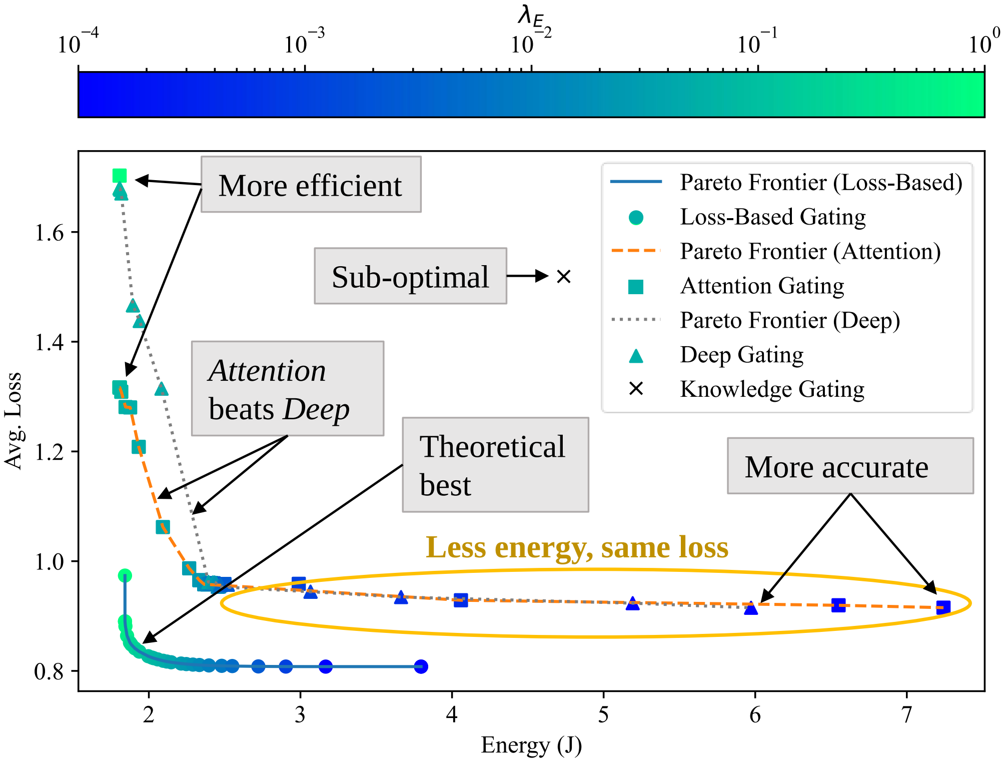


Figure 5.3: System-wide energy consumption vs. object detection loss of different gate modules for varying values of λ_E.

5.4.4 Optimization Analysis

Fig. 5.3 illustrates the trade-off between system-wide energy consumption and model performance for each gate module at different values of λ_E. Both *deep* and *attn* gates present a clear trade-off between performance and energy efficiency as λ_E increases. However, the large flat region along the right side of both Pareto frontiers illustrates how system-wide energy can be reduced significantly with a minimal performance impact. Interestingly, the *attn* gate is more sensitive to changes in λ_E than the *deep* gate, which can be seen in Table 5.1. The results for *loss-based* gating indicate the performance of an optimal gate module and serve as a theoretical upper bound for performance. The results also illustrate why the *knowledge* gate is ineffective, as it cannot minimize either objective well. Overall, the *deep* and *attn* gate reduce energy consumption by over **55%** while maintaining an average loss within 5% of the $\lambda_E = 0$ models.

5.4.5 Scenario-Specific Performance

Fig. 5.4 shows how different driving scenarios affect the energy consumption and performance of different fusion methods. The results show that CARMA can reduce energy consumption below that of early fusion, late fusion, and EcoFusion across all scenarios. Interestingly, our model minimizes energy consumption in the *Snow* scenario by selecting camera branches only throughout the context (C_L, C_R, and $C_L + C_R$). Early fusion is especially weak in the *Fog*, *Rural*, and *Snow* contexts, likely due to its susceptibility to sensor noise. Late fusion, EcoFusion, and CARMA are robust across all scenarios, with *Rural* being the most challenging.

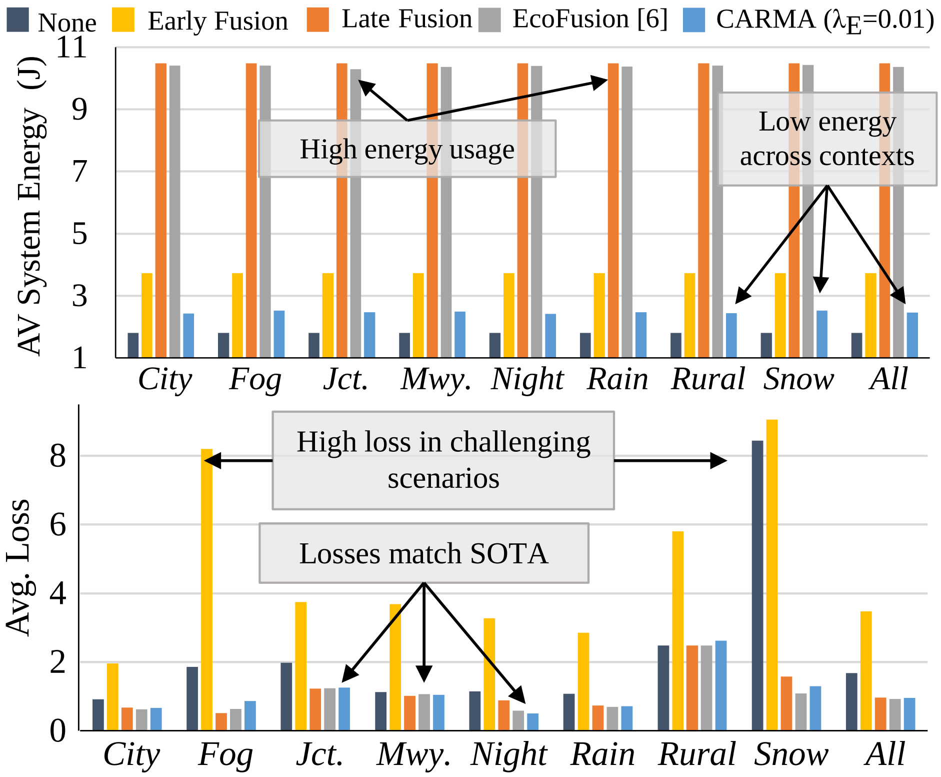


Figure 5.4: Scenario-specific energy usage and object detection loss for: No Fusion (C_R), Early Fusion ($L + C_L + C_R$), Late Fusion ($R + L + C_L + C_R$), EcoFusion ($\lambda_E = 0, attn$), and CARMA ($\lambda_E = 0.01, attn$).

5.5 Summary

In this chapter, we proposed a context-aware sensor fusion approach that uses context to reconfigure the perception model on an FPGA at runtime dynamically. CARMA is capable of switching model computation paths with negligible latency while intermittent context identification, system-wide energy-performance optimization, and sensor clock gating maximize energy savings without compromising performance. Overall, CARMA achieves up to 1.3× speedup and reduces energy consumption by over 73% over leading static and dynamic sensor fusion techniques across complex driving contexts. By using a DPU-based implementation, CARMA can easily be tailored and ported to various FPGA platforms with different amounts of resources. Future works could explore the feasibility of mixing hard-logic and DPU-based solutions for greater energy efficiency.

Chapter 6

Conclusion

6.1 Discussion and Key Findings

Overall, the results support the claim that integrating contextual information into AS models improves scene understanding, improves performance across changing environments, and enables joint energy-performance optimization.

Across the chapters, context modeling is shown to help improve robustness and scene understanding across applications. Chapter 2 shows that explicit modeling of inter-object relationships via scene-graphs can improve scene understanding and transfer learning. Chapter 4 shows how abstract global state estimates drawn from sensor data can be an effective form of context that enable runtime model adaptation for robust cross-scenario performance.

The results also show that context enables adaptations that reduce energy usage without compromising performance. Chapter 4 shows that, since multiple architectural configurations can perform similarly for a given context, an adaptive architecture can selectively choose the most energy-efficient architecture for each context. Chapter 5 illustrates how this joint optimization could be extended to system-wide optimization for greater energy savings. Chapter 3 shows how modeling current network conditions could enable intelligent, low-latency computation offloading from the edge.

Overall, the capability of a model to adapt to changing environments was crucial for optimizing performance, robustness, and energy efficiency. Dynamic ML models can achieve the same caliber of performance as large static ML models with significantly fewer resources, making them more suitable for deployment on edge AS. Chapter 5 demonstrates how a dynamic ML sensor fusion architecture could be mapped to industry-standard FPGAs and be reconfigured within runtime latency constraints. Chapter 3 illustrates how smart low-latency offloading could be implemented using existing network infrastructure with failsafe mechanisms. Currently, most commercially deployed AS use static model designs that follow conventional supervised ML paradigms. However, the benefits of dynamic models cannot be overlooked, so methods for developing, testing, and verifying dynamic models within typical engineering workflows should be studied to enable their broader adoption and utility.

6.2 Limitations and Future Work

Though the methodologies presented in this dissertation provide clear benefits in their respective applications, the scope of these research areas is much broader.

Chapter 2 exemplifies the modeling power and computation efficiency of graph learning. However, graph learning has the potential to enable broader energy savings. Dynamic graph sparsity and hardware optimizations can help reduce computational loads from graph learning models. In addition, Chapter 2 primarily studies graphs extracted using rule-based domain-specific inter-object relations; data-driven graph representations could improve performance by enabling domain adaptation of graphs [40]. An initial study on how data-driven graphs can improve performance and transfer learning is shown in Appendix A. However, data-driven graph extraction and modeling has yet to be studied through the lens of energy efficiency or across other modeling tasks, so this is a clear research direction that can be explored.

Though Chapter 3 demonstrates the benefits of split-architecture computing for end-to-end vehicle control, most current AV pipelines use modular architectures due to the ease of dividing engineering tasks, testing/verifying modules, and integrating domain-knowledge rules and classical algorithms in the complete pipeline. Notably, the perception system is still a significant energy consumer in modular architectures [5], so SAGE can still provide benefits in these cases. Additionally, the offloading algorithm uses a fixed deadline for receiving a response from the cloud; in the real world, deadlines can vary depending on the context of the driving scene and safety guarantees must be met even when offloading. In dangerous situations, the deadline can be lower, while the deadline could be extended on empty roads. To address this limitation, recent works have proposed methods for combining formal safety guarantees with a cloud offloading approach to enable both safety and energy-efficiency[154, 155, 156]. Though these works are promising, combining energy-efficient AS algorithms with formal safety guarantees requires further study.

Chapter 4 explores how a learned abstract context extracted from sensor data via CNNs can enable energy-efficient adaptive sensor fusion. This work also analyzes how fixed domain knowledge context performs in terms of object detection capability and energy efficiency across scenarios. Although the learned context significantly outperforms both static models and domain knowledge context, more advanced context modeling techniques could be employed in future work. Temporal features are often highly relevant in AS, so a spatio-temporal context model could prove more effective. Further, combining domain knowledge and learned contexts could also provide benefits and allow model designers to control adaptation behavior more precisely.

In Chapter 5, energy usage was reduced system-wide by using an FPGA compute platform with sensor clock gating. However, modern edge AS often utilize System-on-Chip (SoC) hardware consisting of multiple heterogeneous computing cores. A promising future research direction would be to study how dynamic architectures can be mapped to different compute cores in a heterogeneous SoC to maximize throughput while minimizing latency and energy consumption across contexts [157]. A combined approach between CARMA and SAGE could also be studied in future work. [158] presents an initial study in this direction and demonstrates clear energy efficiency benefits, proving that there is merit in exploring this research direction.

Overall, most of the methodologies proposed in this dissertation tackled specific AS subproblems (e.g., perception, state estimation) instead of the complete AS pipeline (perception, state estimation, planning, control). The notable exception is Chapter 3, which explored end-to-end control; however, this approach was evaluated using open-loop analysis (i.e., no feedback from control outputs). In the real world, feedback and closed-loop control are essential for effective robot locomotion. In future work, it would be prudent to analyze how closed-loop feedback affects these methodologies' real-world safety, performance, and energy-saving capabilities. In addition, it would be valuable to explore how context can enable adaptation in other modular AS tasks such as localization, path planning, and control. Lastly, most of the proposed methods were evaluated regarding AV constraints and capabilities. Since drones, ground robots, and other small edge AS have unique characteristics but perform fundamentally the same subtasks as AVs, it would be interesting to analyze how these methodologies can be mapped or adapted to other AS with consideration for domain-specific constraints and objectives.

6.3 Final Remarks

Designing energy-efficient autonomous systems remains a complex yet essential research challenge. This dissertation proposes several methodologies that utilize contextual information to improve the performance, energy efficiency, or cross-scenario adaptability of AS. AS energy usage is a growing problem; the scaling up of sensor arrays, compute platforms, and algorithms over time will inevitably lead to more significant resource requirements. System-wide energy usage must be carefully managed to enable large and small AS to benefit society. Since an AS must be able to sense and adapt to changes in its environment to operate effectively, the ML software and hardware systems that power them should also be able to adapt. Though this dissertation presents several solutions to these problems, this research area is broad, and further study is needed along these research directions to enable truly ubiquitous autonomy. Overall, context-aware adaptive AS can enable a safer, more reliable, and more efficient autonomous future.

Bibliography

[1] Marcel Sheeny, Emanuele De Pellegrin, Saptarshi Mukherjee, Alireza Ahrabian, Sen Wang, and Andrew Wallace. RADIATE: A radar dataset for automotive perception. *arXiv preprint arXiv:2010.09076*, 3(4):7, 2020.

[2] David P Watson and David H Scheidt. Autonomous systems. *Johns Hopkins APL technical digest*, 26(4):368–376, 2005.

[3] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020.

[4] Deloitte Development LLC. Global Automotive Consumer Study - North America. Technical report, Deloitte Development LLC, 2020.

[5] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason 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*, pages 751–766, 2018.

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[7] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[8] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018.

[9] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

[10] Fred Lambert. All new Teslas are equipped with NVIDIA's new Drive PX 2 AI platform for self-driving - Electrek. https://electrek.co/2016/10/21/all-new-teslas-are-equipped-with-nvidias-new-drive-px-2-ai-platform-for-self-driving, Oct 2016.

[11] Kyle Hyatt. Baidu unveils its camera-based Apollo Lite self-driving suite. *Roadshow*, Jun 2019.

[12] National Transportation Safety Board. Collision Between a Sport Utility Vehicle Operating With Partial Driving Automation and a Crash Attenuator. Technical Report NTSB/HAR-20/01, National Transportation Safety Board, 2020.

[13] National Transportation Safety Board. Collision Between Car Operating with Partial Driving Automation and Truck-Tractor Semitrailer. Technical Report NTSB/HAB-20/01, National Transportation Safety Board, 2020.

[14] National Transportation Safety Board. Collision between vehicle controlled by developmental automated driving system and pedestrian. Technical Report NTSB/HAR-19/03, National Transportation Safety Board, 2019.

[15] Sebastian Höfer, Kostas Bekris, Ankur Handa, Juan Camilo Gamboa, Melissa Mozifian, Florian Golemo, Chris Atkeson, Dieter Fox, Ken Goldberg, John Leonard, et al. Sim2real in robotics and automation: Applications and challenges. *IEEE transactions on automation science and engineering*, 18(2):398–400, 2021.

[16] Nate Oh. NVIDIA Announces Drive PX Pegasus at GTC Europe 2017: Level 5 Self-Driving Hardware, Feat. Post-Volta GPUs. *AnandTech*, Oct 2017.

[17] Sam Abuelsamid. Nvidia Cranks Up And Turns Down Its Drive AGX Orin Computers. *Forbes*, Jun 2020.

[18] Navtech Radar. Navtech CTS Series, May 2021.

[19] Velodyne Lidar. Velodyne HDL-32e Datasheet, May 2021.

[20] X He, HC Kim, R Ma, TJ Wallington, et al. Energy consumption simulation for connected and automated vehicles: Eco-driving benefits versus automation loads. *SAE Int. J. of Connected and Autonomous Vehicles*, 6, 2022.

[21] Spot Specifications, April 2022. [Online; accessed 16. Apr. 2023].

[22] DJI Air 2S - Specs - DJI, April 2023. [Online; accessed 16. Apr. 2023].

[23] Ivan Beretta, Francisco Rincon, Nadia Khaled, Paolo Roberto Grassi, Vincenzo Rana, and David Atienza. Design exploration of energy-performance trade-offs for wireless sensor networks. In *Proceedings of the 49th Annual Design Automation Conference*, pages 1043–1048, 2012.

[24] Shreyas Sen. Context-aware energy-efficient communication for iot sensor nodes. In *53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2016.

[25] Korosh Vatanparvar and Mohammad Abdullah Al Faruque. Battery lifetime-aware automotive climate control for electric vehicles. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[26] Korosh Vatanparvar, Jiang Wan, and Mohammad Abdullah Al Faruque. Battery-aware energy-optimal electric vehicle driving management. In *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 353–358. IEEE, 2015.

[27] Korosh Vatanparvar, Sina Faezi, Igor Burago, Marco Levorato, and Mohammad Abdullah Al Faruque. Extended range electric vehicle with driving behavior estimation in energy management. *IEEE transactions on Smart Grid*, 10(3):2959–2968, 2018.

[28] Korosh Vatanparvar and Mohammad Abdullah Al Faruque. Eco-friendly automotive climate control and navigation system for electric vehicles. In *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*, pages 1–10. IEEE, 2016.

[29] Korosh Vatanparvar and Mohammad Abdullah Al Faruque. Design and analysis of battery-aware automotive climate control for electric vehicles. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(4):1–22, 2018.

[30] Korosh Vatanparvar. *Reliable and Energy Efficient Battery-Powered Cyber-Physical Systems*. PhD thesis, University of California, Irvine, 2018.

[31] Korosh Vatanpavar and Mohammad Abdullah Al Faruque. Acqua: Adaptive and cooperative quality-aware control for automotive cyber-physical systems. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 193–200. IEEE, 2017.

[32] Peng-Yong Kong. Computation and sensor offloading for cloud-based infrastructure-assisted autonomous vehicles. *IEEE Systems Journal*, 14(3):3360–3370, 2020.

[33] K. Samal, M. Wolf, and S. Mukhopadhyay. Attention-based activation pruning to reduce data movement in real-time ai: A case-study on local motion planning in autonomous vehicles. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(3):306–319, 2020.

[34] Mohammad Abdullah Al Faruque and Korosh Vatanparvar. Energy management-as-a-service over fog computing platform. *IEEE internet of things journal*, 3(2):161–169, 2015.

[35] Mohanad Odema, Nafiul Rashid, Berken Utku Demirel, and Mohammad Abdullah Al Faruque. Lens: Layer distribution enabled neural architecture search in edge-cloud hierarchies. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021.

[36] Gyorgy Buzsaki. *Rhythms of the Brain*. Oxford university press, 2006.

[37] Yiping Gong, Zhifeng Xiao, Xiaowei Tan, Haigang Sui, Chuan Xu, Haiwang Duan, and Deren Li. Context-aware convolutional neural network for object detection in vhr remote sensing imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 58(1):34–44, 2019.

[38] Shih-Yuan Yu, Arnav Vaibhav Malawade, Deepan Muthirayan, Pramod P Khargonekar, and Mohammad Abdullah Al Faruque. Scene-graph augmented data-driven risk assessment of autonomous vehicle decisions. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[39] Arnav Vaibhav Malawade, Shih-Yuan Yu, Brandon Hsu, Deepan Muthirayan, Pramod P Khargonekar, and Mohammad Abdullah Al Faruque. Spatiotemporal scene-graph embedding for autonomous vehicle collision prediction. *IEEE Internet of Things Journal*, 9(12):9379–9388, 2022.

[40] Arnav Vaibhav Malawade, Shih-Yuan Yu, Junyao Wang, and Mohammad Abdullah Al Faruque. Rs2g: Data-driven scene-graph extraction and embedding for robust autonomous perception and scenario understanding. *arXiv preprint arXiv:2304.08600*, 2023.

[41] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[42] Nikita Japuria, Golnaz Habibi, and Jonathan How. Casnsc: A context-based approach for accurate pedestrian motion prediction at intersections. In *Conference and Workshop on Neural Information Processing Systems (NEURIPS)*, 2017.

[43] Arnav Vaibhav Malawade, Trier Mortlock, and Mohammad Abdullah Al Faruque. Ecofusion: Energy-aware adaptive sensor fusion for efficient autonomous vehicle perception. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 481–486, 2022.

[44] Arnav Malawade, Mohanad Odema, Sebastien Lajeunesse-DeGroot, and Mohammad Abdullah Al Faruque. Sage: A split-architecture methodology for efficient end-to-end autonomous vehicle control. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s):1–22, 2021.

[45] Federico Bartoli, Giuseppe Lisanti, Lamberto Ballan, and Alberto Del Bimbo. Context-aware trajectory prediction. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 1941–1946. IEEE, 2018.

[46] Radoslav Ivanov, James Weimer, and Insup Lee. Context-aware detection in medical cyber-physical systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 232–241. IEEE, 2018.

[47] Nafiul Rashid, Trier Mortlock, and Mohammad Abdullah Al Faruque. Self-care: Selective fusion with context-aware low-power edge computing for stress detection. In *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 49–52. IEEE, 2022.

[48] Nafiul Rashid, Trier Mortlock, and Mohammad Abdullah Al Faruque. Stress detection using context-aware sensor fusion from wearable devices. *IEEE Internet of Things Journal*, 2023.

[49] Nafiul Rashid. *Efficient Digital Health Solutions Using Wearable Devices*. PhD thesis, University of California, Irvine, 2023.

[50] Nafiul Rashid, Manik Dautta, Peter Tseng, and Mohammad Abdullah Al Faruque. Hear: Fog-enabled energy-aware online human eating activity recognition. *IEEE Internet of Things Journal*, 8(2):860–868, 2020.

[51] Saehanseul Yi, Tae-Wook Kim, Jong-Chan Kim, and Nikil Dutt. Energy-efficient adaptive system reconfiguration for dynamic deadlines in autonomous driving. In *2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 96–104. IEEE, 2021.

[52] Sanghoon Lee, Dongkyu Lee, Pyung Choi, and Daejin Park. Accuracy–power controllable lidar sensor system with 3D object recognition for autonomous vehicle. *Sensors*, 20(19):5706, 2020.

[53] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[54] Nafiul Rashid, Berken Utku Demirel, and Mohammad Abdullah Al Faruque. Ahar: Adaptive cnn for energy-efficient human activity recognition in low-power edge devices. *IEEE Internet of Things Journal*, 9(15):13041–13051, 2022.

[55] Alexandra S Mueller, Jessica B Cicchino, and David S Zuby. What humanlike errors do autonomous vehicles need to avoid to maximize safety? *Journal of Safety Research*, 2020.

[56] Brandon Schoettle and Michael Sivak. A preliminary analysis of real-world crashes involving self-driving vehicles. Technical Report UMTRI-2015-34, University of Michigan Transportation Research Institute, 2015.

[57] Chengcheng Xu, Zijian Ding, Chen Wang, and Zhibin Li. Statistical analysis of the patterns and characteristics of connected and autonomous vehicle involved crashes. *Journal of safety research*, 71:41–47, 2019.

[58] Ekim Yurtsever, Yongkang Liu, Jacob Lambert, Chiyomi Miyajima, Eijiro Takeuchi, Kazuya Takeda, and John HL Hansen. Risky action recognition in lane change video clips using deep spatiotemporal networks with segmentation mask transfer. *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3100–3107, 2019.

[59] Chongben Tao, Haotian He, Fenglei Xu, and Jiecheng Cao. Stereo priori rcnn based car detection on point level for autonomous driving. *Knowledge-Based Systems*, 229:107346, 2021.

[60] Degui Xiao, Xuefeng Yang, Jianfang Li, and Merabtene Islam. Attention deep neural network for lane marking detection. *Knowledge-Based Systems*, 194:105584, 2020.

[61] Sebastian Sontges, Markus Koschi, and Matthias Althoff. Worst-case analysis of the time-to-react using reachable sets. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1891–1897. IEEE, 2018.

[62] David Nistér, Hon-Leung Lee, Julia Ng, and Yizhou Wang. The safety force field. *NVIDIA White Paper*, 2019.

[63] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[64] Sravan Mylavarapu, Mahtab Sandhu, Priyesh Vijayan, K Madhava Krishna, Balaraman Ravindran, and Anoop Namboodiri. Towards accurate vehicle behaviour classification with multi-relational graph convolutional networks. *arXiv preprint arXiv:2002.00786*, 2020.

[65] Chengxi Li, Yue Meng, Stanley H Chan, and Yi-Ting Chen. Learning 3d-aware egocentric spatial-temporal interaction via graph convolutional networks. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8418–8424, 2020.

[66] Lars Kunze, Tom Bruls, Tarlan Suleymanov, and Paul Newman. Reading between the lanes: Road layout reconstruction from partially segmented scenes. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 401–408, 2018.

[67] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[68] Albert-László Barabási. Network science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1987):20120375, 2013.

[69] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[70] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[71] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[72] Shih-Yuan Yu. *Machine Learning on Graphs for Robust and Secure Embedded Systems.* PhD thesis, University of California, Irvine, 2023.

[73] Chengxi Li, Yue Meng, Stanley H Chan, and Yi-Ting Chen. Learning 3d-aware egocentric spatial-temporal interaction via graph convolutional networks. *arXiv preprint arXiv:1909.09272*, 2019.

[74] Yafu Tian, Alexander Carballo, Ruifeng Li, and Kazuya Takeda. Road scene graph: A semantic graph-based scene representation dataset for intelligent vehicles. *arXiv preprint arXiv:2011.13588*, 2020.

[75] Xiaoming Liu, Yu Lan, Yadong Zhou, Chao Shen, and Xiaohong Guan. A real-time explainable traffic collision inference framework based on probabilistic graph theory. *Knowledge-Based Systems*, 212:106442, 2021.

[76] Highway Loss Data Institute. Volvo collision avoidance features: initial results. *Highway Loss Data Institute Bulletin*, 29(5), 2012.

[77] John Dahl, Gabriel Rodrigues de Campos, Claes Olsson, and Jonas Fredriksson. Collision avoidance: A literature review on threat-assessment techniques. *IEEE Transactions on Intelligent Vehicles*, 4(1):101–113, 2018.

[78] Jonas Nilsson, Anders CE Ödblom, and Jonas Fredriksson. Worst-case analysis of automotive collision avoidance systems. *IEEE Transactions on Vehicular Technology*, 65(4):1899–1911, 2015.

[79] Matthias Althoff, Olaf Stursberg, and Martin Buss. Model-based probabilistic collision detection in autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 10(2):299–310, 2009.

[80] Yijing Wang, Zhengxuan Liu, Zhiqiang Zuo, Zheng Li, Li Wang, and Xiaoyuan Luo. Trajectory planning and safety assessment of autonomous vehicles based on motion prediction and model predictive control. *IEEE Transactions on Vehicular Technology*, 68(9):8546–8556, 2019.

[81] Lijun Zhang, Wei Xiao, Zhuang Zhang, and Dejian Meng. Surrounding vehicles motion prediction for risk assessment and motion planning of autonomous vehicle in highway scenarios. *IEEE Access*, 8:209356–209376, 2020.

[82] Xin Wang, Jing Liu, Tie Qiu, Chaoxu Mu, Chen Chen, and Pan Zhou. A real-time collision prediction mechanism with deep learning for intelligent transportation system. *IEEE transactions on vehicular technology*, 69(9):9497–9508, 2020.

[83] Mark Strickland, Georgios Fainekos, and Heni Ben Amor. Deep predictive models for collision risk assessment in autonomous driving. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, 2018.

[84] Bahar Asgari, Ramyad Hadidi, Nima Shoghi Ghaleshahi, and Hyesoon Kim. Pisces: Power-aware implementation of slam by customizing efficient sparse algebra. *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[85] Sabur Baidya, Yu-Jen Ku, Hengyu Zhao, Jishen Zhao, and Sujit Dey. Vehicular and edge computing for emerging connected and autonomous vehicle applications. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[86] Chao Huang, Shichao Xu, Zhilu Wang, Shuyue Lan, Wenchao Li, and Qi Zhu. Opportunistic intermittent control with safety guarantees for autonomous systems. In *DAC '20*, pages 1–6. IEEE, 2020.

[87] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph r-cnn for scene graph generation. *Proceedings of the European conference on computer vision (ECCV)*, pages 670–685, 2018.

[88] Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5410–5419, 2017.

[89] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, 2017.

[90] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. *European Semantic Web Conference*, pages 593–607, 2018.

[91] Hongyang Gao and Shuiwang Ji. Graph u-nets. *arXiv preprint arXiv:1905.05178*, 2019.

[92] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082*, 2019.

[93] Arnav Vaibhav Malawade, Shih-Yuan Yu, Brandon Hsu, Harsimrat Kaeley, Anurag Karra, and Mohammad Abdullah Al Faruque. Roadscene2vec: A tool for extracting and embedding road scene-graphs. *Knowledge-Based Systems*, 242:108245, 2022.

[94] Vasili Ramanishka, Yi-Ting Chen, Teruhisa Misu, and Kate Saenko. Toward driving scene understanding: A dataset for learning driver behavior and causal reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7699–7707, 2018.

[95] Yu Yao, Xizi Wang, Mingze Xu, Zelin Pu, Ella Atkins, and David Crandall. When, where, and what? a new dataset for anomaly detection in driving videos. *arXiv preprint arXiv:2004.03044*, 2020.

[96] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[97] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):6, 2020.

[98] Jingyun Feng, Zhi Liu, Celimuge Wu, and Yusheng Ji. Mobile edge computing for the internet of vehicles: Offloading framework and job scheduling. *IEEE vehicular technology magazine*, 14(1):28–36, 2018.

[99] Kyle Hyatt. Argo gives its self-driving vehicle hardware a big upgrade. *Roadshow*, Jan 2021.

[100] Apostolos Papathanassiou and Alexey Khoryaev. Cellular v2x as the essential enabler of superior global connected transportation services. *IEEE 5G Tech Focus*, 1(2):1–2, 2017.

[101] Stephan Eichler. Performance evaluation of the ieee 802.11 p wave communication standard. In *2007 IEEE 66th Vehicular Technology Conference*, pages 2199–2203. IEEE, 2007.

[102] Mingyue Cui, Shipeng Zhong, Boyang Li, Xu Chen, and Kai Huang. Offloading autonomous driving services via edge computing. *IEEE Internet of Things Journal*, 7(10):10535–10547, 2020.

[103] Kengo Sasaki, Naoya Suzuki, Satoshi Makido, and Akihiro Nakao. Vehicle control system coordinated between cloud and mobile edge computing. In *2016 55th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pages 1122–1127. IEEE, 2016.

[104] Ke Zhang, Yuming Mao, Supeng Leng, Sabita Maharjan, and Yan Zhang. Optimal delay constrained offloading for vehicular edge computing networks. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2017.

[105] Ke Zhang, Yuming Mao, Supeng Leng, Quanxin Zhao, Longjiang Li, Xin Peng, Li Pan, Sabita Maharjan, and Yan Zhang. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE access*, 4:5896–5907, 2016.

[106] Y. Matsubara, D. Callegaro, S. Baidya, M. Levorato, and S. Singh. Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems. *IEEE Access*, 8:212177–212193, 2020.

[107] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778. IEEE Computer Society, 2016.

[108] Simon Hecker, Dengxin Dai, and Luc Van Gool. End-to-end learning of driving models with surround-view cameras and route planners. *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 435–453, 2018.

[109] Alejandro González, Zhijie Fang, Yainuvis Socarras, Joan Serrat, David Vázquez, Jiaolong Xu, and Antonio M. López. Pedestrian detection at day/night time with visible and fir cameras: A comparison. *Sensors*, 16(6), 2016.

[110] Young-Duk Kim, Guk-Jin Son, Chan-Ho Song, and Hee-Kang Kim. On the deployment and noise filtering of vehicular radar application for detection enhancement in roads and tunnels. *Sensors*, 18(3), 2018.

[111] Xiaolu Li, Bingwei Yang, Xinhao Xie, Duan Li, and Lijun Xu. Influence of waveform characteristics on lidar ranging accuracy and precision. *Sensors*, 18(4), 2018.

[112] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *International Conference on Robotics and Automation (ICRA)*, 2018.

[113] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[114] Ardi Tampuu, Tambet Matiisen, Maksym Semikin, Dmytro Fishman, and Naveed Muhammad. A survey of end-to-end driving: Architectures and training methods. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[115] Xiufeng Xie and Kyu-Han Kim. Source compression with bounded dnn perception loss for iot edge computer vision. In *The 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, New York, NY, USA, 2019. Association for Computing Machinery.

[116] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, page 615–629, 2017.

[117] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer Singh. Distilled split deep neural networks for edge-assisted real-time systems. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, HotEdgeVideo'19, page 21–26, New York, NY, USA, 2019. Association for Computing Machinery.

[118] Yoshitomo Matsubara and Marco Levorato. Neural compression and filtering for edge-assisted real-time object detection in challenged networks, 2020.

[119] Lei Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2654–2662, Cambridge, MA, USA, 2014. MIT Press.

[120] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[121] Gregor Urban, Krzysztof J. Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. Do deep convolutional nets really need to be deep and convolutional?, 2017.

[122] Jang Hyun Cho and Bharath Hariharan. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[123] Junxian Huang et al. A close examination of performance and power characteristics of 4G lte networks. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, page 225–238, 2012.

[124] Agbotiname Lucky Imoize, Kehinde Orolu, and Aderemi Aaron-Anthony Atayero. Analysis of key performance indicators of a 4g lte network based on experimental data obtained from a densely populated smart city. *Data in brief*, 29:105304, 2020.

[125] ARM. Arm npu ethos-77, 2021.

[126] Felipe Codevilla, Antonio M Lopez, Vladlen Koltun, and Alexey Dosovitskiy. On offline evaluation of vision-based driving models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 236–251, 2018.

[127] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9329–9338, 2019.

[128] M. Khayyat, I. A. Elgendy, A. Muthanna, A. S. Alshahrani, S. Alharbi, and A. Koucheryavy. Advanced deep learning-based computational offloading for multilevel vehicular edge-cloud computing networks. *IEEE Access*, 8:137052–137062, 2020.

[129] Vineet Gokhale, Gerardo Moyers Barrera, and R Venkatesha Prasad. Feel: fast, energy-efficient localization for autonomous indoor vehicles. *arXiv preprint arXiv:2102.00702*, 2021.

[130] Dieter Balemans, Wim Casteels, Simon Vanneste, Jens de Hoog, Siegfried Mercelis, and Peter Hellinckx. Resource efficient sensor fusion by knowledge-based network pruning. *Internet of Things*, 11:100231, 2020.

[131] Arnav Vaibhav Malawade, Trier Mortlock, and Mohammad Abdullah Al Faruque. Hydrafusion: Context-aware selective sensor fusion for robust and efficient autonomous vehicle perception. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pages 68–79. IEEE, 2022.

[132] Jared A Baxter, Daniel A Merced, Daniel J Costinett, Leon M Tolbert, and Burak Ozpineci. Review of electrical architectures and power requirements for automated vehicles. In *2018 IEEE Transportation Electrification Conference and Expo (ITEC)*, pages 944–949. IEEE, 2018.

[133] Justin M Bradley and Ella M Atkins. Optimization and control of cyber-physical vehicle systems. *Sensors*, 15(9):23020–23049, 2015.

[134] Changhao Chen, Stefano Rosa, Yishu Miao, Chris Xiaoxuan Lu, Wei Wu, Andrew Markham, and Niki Trigoni. Selective sensor fusion for neural visual-inertial odometry. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10542–10551, 2019.

[135] Ravi Teja Mullapudi, William R Mark, Noam Shazeer, and Kayvon Fatahalian. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8080–8089, 2018.

[136] Boyu Zhang, Azadeh Davoodi, and Yu H. Hu. Exploring energy and accuracy tradeoff in structure simplification of trained deep neural networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(4):836–848, 2018.

[137] Hokchhay Tann, Soheil Hashemi, R Iris Bahar, and Sherief Reda. Runtime configurable deep neural networks for energy-accuracy trade-off. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 1–10. IEEE, 2016.

[138] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015.

[139] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *International Conference on Machine Learning*, pages 7354–7363. PMLR, 2019.

[140] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[141] Roman Solovyev, Weimin Wang, and Tatiana Gabruseva. Weighted boxes fusion: Ensembling boxes from different object detection models. *Image and Vision Computing*, 107:104117, 2021.

[142] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[143] Stereolabs. ZED Camera and SDK Overview, May 2021.

[144] Xilinx. Vitis AI User Guide (UG1414), June 2022.

[145] Hasan Irmak, Daniel Ziener, and Nikolaos Alachiotis. Increasing flexibility of fpga-based cnn accelerators with dynamic partial reconfiguration. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, pages 306–311. IEEE, 2021.

[146] Eman Youssef, Hamed A Elsemary, Magdy A El-Moursy, Ahmed Khattab, and Hassan Mostafa. Energy adaptive convolution neural network using dynamic partial reconfiguration. In *MWSCAS 2020*, pages 325–328. IEEE, 2020.

[147] Abdelrahman S. Hussein et al. Implementation of a DPU-based intelligent thermal imaging hardware accelerator on FPGA. *Electronics*, 2022.

[148] Zafar Takhirov, Joseph Wang, Venkatesh Saligrama, and Ajay Joshi. Energy-efficient adaptive classifier design for mobile systems. In *ISLPED '16*, page 52–57. Association for Computing Machinery, 2016.

[149] Kruttidipta Samal, Hemant Kumawat, Priyabrata Saha, Marilyn Wolf, and Saibal Mukhopadhyay. Task-driven rgb-lidar fusion for object tracking in resource-efficient autonomous system. *IEEE Transactions on Intelligent Vehicles*, 7(1):102–112, 2021.

[150] Ranjita Dash and Harish J Palanthandalam-Madapusi. Intermittent control in autonomous vehicle steering control and lane keeping. In *5th International Conference of The Robotics Society*, pages 1–6, 2021.

[151] Donkyu Baek, Yukai Chen, Alberto Bocca, Alberto Macii, Enrico Macii, and Massimo Poncino. Battery-aware energy model of drone delivery tasks. In *ISLPED '18*, 2018.

[152] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[153] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[154] Mohanad Odema, James Ferlez, Yasser Shoukry, and Mohammad Abdullah Al Faruque. Seo: Safety-aware energy optimization framework for multi-sensor neural controllers at the edge. *arXiv preprint arXiv:2302.12493*, 2023.

[155] Mohanad Odema, James Ferlez, Goli Vaisi, Yasser Shoukry, and Mohammad Abdullah Al Faruque. Energyshield: Provably-safe offloading of neural network controllers for energy efficiency. *arXiv preprint arXiv:2302.06572*, 2023.

[156] Mohanad Odema, Luke Chen, Marco Levorato, and Mohammad Abdullah Al Faruque. Testudo: Collaborative intelligence for latency-critical autonomous systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[157] Halima Bouzidi, Mohanad Odema, Hamza Ouarnoughi, Smail Niar, and Mohammad Abdullah Al Faruque. Map-and-conquer: Energy-efficient mapping of dynamic neural nets onto heterogeneous mpsocs. *arXiv preprint arXiv:2302.12926*, 2023.

[158] Luke Chen, Mohanad Odema, and Mohammad Abdullah Al Faruque. Romanus: Robust task offloading in modular multi-sensor autonomous driving systems. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–8, 2022.

[159] GB Grayson, G Maycock, JA Groeger, SM Hammond, and DT Field. Risk, hazard perception and perceived control. *TRL Report TRL560. TRL Ltd., Crowthorne, UK*, 2003.

[160] Li Wang, Ziying Song, Xinyu Zhang, Chenfei Wang, Guoxin Zhang, Lei Zhu, Jun Li, and Huaping Liu. Sat-gcn: Self-attention graph convolutional network-based 3d object detection for autonomous driving. *Knowledge-Based Systems*, 259:110080, 2023.

[161] Sergio Casas, Cole Gulino, Renjie Liao, and Raquel Urtasun. Spagnn: Spatially-aware graph neural networks for relational behavior forecasting from sensor data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9491–9497. IEEE, 2020.

[162] Nachiket Deo, Eric Wolff, and Oscar Beijbom. Multimodal trajectory prediction conditioned on lane-graph traversals. In *Conference on Robot Learning*, pages 203–212. PMLR, 2022.

[163] Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data. In *European Conference on Computer Vision*, pages 683–700. Springer, 2020.

[164] Pawit Kochakarn, Daniele De Martini, Daniel Omeiza, and Lars Kunze. Explainable action prediction through self-supervision on scene graphs. *arXiv preprint arXiv:2302.03477*, 2023.

[165] Sravan Mylavarapu, Mahtab Sandhu, Priyesh Vijayan, K Madhava Krishna, Balaraman Ravindran, and Anoop Namboodiri. Understanding dynamic scenes using graph convolution networks. *arXiv preprint arXiv:2005.04437*, 2020.

[166] Hang Xu, Linpu Fang, Xiaodan Liang, Wenxiong Kang, and Zhenguo Li. Universal-rcnn: Universal object detector via transferable graph r-cnn. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12492–12499, 2020.

[167] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

[168] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[169] Ethan Zhang, Sion Pizzi, and Neda Masoud. A learning-based method for predicting heterogeneous traffic agent trajectories: implications for transfer learning. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 1853–1858. IEEE, 2021.

[170] Nick Lamm, Shashank Jaiprakash, Malavika Srikanth, and Iddo Drori. Vehicle trajectory prediction by transfer learning of semi-supervised models. *arXiv preprint arXiv:2007.06781*, 2020.

[171] Ana I Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso García, and Davide Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5419–5427, 2018.

[172] Shivam Akhauri, Laura Zheng, Tom Goldstein, and Ming Lin. Improving generalization of transfer learning across domains using spatio-temporal features in autonomous driving. *arXiv preprint arXiv:2103.08116*, 2021.

[173] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[174] Jiawei Han, Micheline Kamber, Jian Pei, et al. Getting to know your data. In *Data mining*, volume 2, pages 39–82. Morgan Kaufmann Boston, MA, USA, 2012.

[175] Sina Faezi, Sujit Rokka Chhetri, Arnav Vaibhav Malawade, John Charles Chaput, William Grover, Philip Brisk, and Mohammad Abdullah Al Faruque. Oligo-snoop: a non-invasive side channel attack against dna synthesis machines. In *Network and Distributed Systems Security (NDSS) Symposium 2019*, 2019.

[176] Sina Faezi. *Data-Driven Modeling and Analysis for Trustworthy Cyber-Physical Systems*. PhD thesis, University of California, Irvine, 2021.

[177] Sina Faezi, Sujit Rokka Chhetri, Arnav Vaibhav Malawade, John Charles Chaput, William Grover, Philip Brisk, and Mohammad Abdullah Al Faruque. Acoustic side channel attack against dna synthesis machines. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 186–187. IEEE, 2020.

[178] Sujit Rokka Chhetri, Sina Faezi, Nafiul Rashid, and Mohammad Abdullah Al Faruque. Manufacturing supply chain and product lifecycle security in the era of industry 4.0. *Journal of Hardware and Systems Security*, 2:51–68, 2018.

[179] Sujit Rokka Chhetri, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. Kcad: kinetic cyber-attack detection method for cyber-physical additive manufacturing systems. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2016.

[180] Sujit Rokka Chhetri and Mohammad Abdullah Al Faruque. Side channels of cyber-physical systems: Case study in additive manufacturing. *IEEE Design & Test*, 34(4):18–25, 2017.

[181] Sujit Rokka Chhetri, Sina Faezi, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. Quilt: quality inference from living digital twins in iot-enabled manufacturing systems. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, pages 237–248, 2019.

[182] Sujit Rokka Chhetri, Anomadarshi Barua, Sina Faezi, Francesco Regazzoni, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. Tool of spies: Leaking

your ip by altering the 3d printer compiler. *IEEE Transactions on Dependable and Secure Computing*, 18(2):667–678, 2019.

[183] Shih-Yuan Yu, Arnav Vaibhav Malawade, Sujit Rokka Chhetri, and Mohammad Abdullah Al Faruque. Sabotage attack detection for additive manufacturing systems. *IEEE Access*, 8:27218–27231, 2020.

[184] Sujit Rokka Chhetri. *Data-Driven Modeling of Cyber-Physical Systems using Side-Channel Analysis*. PhD thesis, University of California, Irvine, 2019.

[185] A. V. Malawade, N. D. Costa, D. Muthirayan, P. P. Khargonekar, and M. A. Al Faruque. Neuroscience-inspired algorithms for the predictive maintenance of manufacturing systems. *IEEE Transactions on Industrial Informatics*, pages 1–1, 2021.

[186] Anthony Lopez, Arnav Vaibhav Malawade, Mohammad Abdullah Al Faruque, Srivalli Boddupalli, and Sandip Ray. Security of emergent automotive systems: A tutorial introduction and perspectives on practice. *IEEE Design & Test*, 36(6):10–38, 2019.

[187] Anthony Bahadir Lopez. *Security Modeling and Analysis for Intelligent Transportation Systems*. PhD thesis, University of California, Irvine, 2020.

[188] Jiang Wan, Anthony Lopez, and Mohammad Abdullah Al Faruque. Physical layer key generation: Securing wireless communication in automotive cyber-physical systems. *ACM Trans. Cyber-Phys. Syst.*, 3(2), oct 2018.

[189] Kelvin Phan, Arnav Malawade, Anthony Lopez, Anomadarshi Barua, Preston Rogers, Ken Tran, and Mohammad Al Faruque. Poster: Physical layer key generation protocol for secure v2x communication architecture. *Databases*, page 216227, 2017.

[190] Soheyb Ribouh, Kelvin Phan, Arnav Vaibhav Malawade, Yassin Elhillali, Atika Rivenq, and Mohammad Abdullah Al Faruque. Channel state information-based cryptographic key generation for intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 22(12):7496–7507, 2020.

Appendix A

Data-Driven Scene-Graph Extraction and Embedding for Robust Scene Understanding

A.1 Overview

This appendix discusses my recent work which proposes using data-driven scene-graph extraction and embedding for AS scene understanding applications. Chapter 2 demonstrated how a rule-based scene-graph extraction and embedding approach enables DL algorithms to explicitly model inter-object relationships for scene understanding tasks such as collision prediction. The proposed method uses domain-knowledge rules to define when to build graph edges for each relation type. In contrast, this appendix proposes a data-driven approach for learning the rules for building relations and demonstrate how it can improve performance over that of rule-based graph extraction across domains. The approach is applied to the problem of subjective risk assessment following [38].

A.2 Introduction

As discussed in Chapter 2, humans naturally model interactions between agents (e.g., humans, animals, dynamic obstacles) [41] to reason about diverse driving scenarios. As a human driver's subjective risk assessment is inversely related to the risk of traffic accidents [159], many works attempt to model the human driving experience to improve the safety of autonomous driving systems further. Under this circumstance, the effectiveness of understanding the driving scenes becomes critical for enhancing the robustness of AVs. Generally speaking, AV tasks that involve understanding and interacting with human agents are the most challenging tasks to solve. Negotiating intersections, crosswalks, and stop signs with human-driven vehicles and pedestrians proves challenging as an AV must understand the intent and behavior of the other agents in the environment. These challenges become even more difficult to overcome, given that AVs are typically designed, trained, and tested using simulations and curated datasets, while real-world scenarios are much more dynamic, unpredictable, and complex. These challenges motivate the need for generalizable models that can effectively transfer knowledge gained from simulation environments to real-world driving conditions without major performance degradation.

A.2.1 Research Challenges

To date, many works leverage *Deep Learning* (DL) to model human driving capabilities, attempting to generalize driving knowledge gained from training datasets to real-world driving scenarios. For example, *Convolution Neural Networks* (CNNs) or physics-based models have been used for this purpose [160, 78]. However, they can fail to account for high-level semantic scene information, thus underperforming in complex or novel scenarios. Specifically, when developing these approaches, the interactions between road users and the environmental factors in the scene (e.g., traffic signals may affect behavior) may not be considered [38]. To address this gap, graph-based modeling utilizing *Graph Learning* (GL) have been proposed over the years [161, 162, 93, 65, 64, 163]; While CNN-based approaches focus on extracting visual features from the road scenes, GL-based ones extract the scene-graph representations of road scenarios, which has proven more effective at explicitly modeling the interactions between these visual features at a higher level [38, 39]. The approach proposed in Chapter 2 supports this point, since it outperforms the state-of-the-art DL-based method at collision prediction. Nevertheless, existing graph-based approaches typically require expert knowledge to design the graphical structure (e.g., rule-based distance relations in [162], [163], and [38]; domain knowledge of road topology in [66]; rule-based directional relations in [64, 38]). This constraint results in a rigid graph construction approach that may not be flexible enough to generalize to new domains and real-world scenarios absent from the training data.

For data-driven approaches, many existing solutions can improve the ability of a model to generalize across domains, i.e., can enhance the model's *robustness*. One straightforward approach is to train DL models on massive labeled driving datasets. However, these datasets are often biased toward everyday driving situations and lack a diversity of edge cases correlated with higher safety risks. These limitations can lead to poor performance in rare scenarios and cause an otherwise highly functional system to maneuver the vehicle into a dangerous situation. In addition, massive datasets can be prohibitively expensive as it incurs significant model training and data storage costs. Alternatively, one can improve the generalization of models using large DL models to process driving data, assuming that large enough models can better capture the complexity and nuance of road scenarios and thus perform better in complex driving situations. However, AVs are real-time systems and edge devices with constrained computational capacity and limited onboard energy storage. As a result, infinitely large models cannot be used, and model size is limited by the ability of the hardware to execute the model in real time. In summary, the key research challenge in developing a data-driven AV system is to support these capabilities.

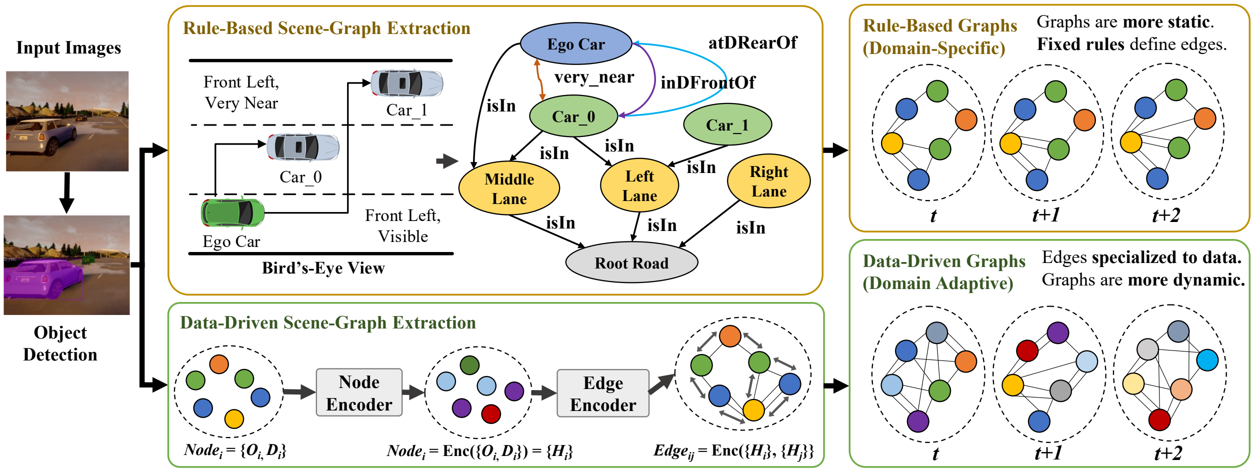


Figure A.1: The differences between the scene-graphs generated by RS2G and those extracted with a rule-based method for a driving scene. The rule-based graphs are more rigid and can only excel at the domain for which the rules are designed, making them *domain-specific*. In contrast, the data-driven graphs can specialize to the input data, resulting in more dynamic *domain-adaptive* graph representations.

- The system has to generalize to a wide range of complex driving scenarios without the need for large datasets,
- The system should learn to explicitly model and understand interactions between agents and the driving environment,
- The system can effectively transfer knowledge gained from simulation and training environments to real-world use cases.

A.2.2 Novel Contributions

Prior work has shown that graph representations of road scenes can improve modeling capability, data efficiency, and transfer learning at AV safety-related tasks. Specifically, [38] and [39] have shown the benefits in risk assessment and collision prediction, both of which are important for tasks such as driver control hand-off, *Advanced Driver Assistance Systems* (ADAS), and dynamic driving profile adjustment. However, these methods require expertise to define topology and domain-specific rules for graph construction, thus being limited to generalizing and adapting to out-of-domain scenarios, a similar limitation faced by expert systems in the past. To address this limitation, we propose **RoadScene2Graph (RS2G)**: a data-driven graph extraction and modeling approach that learns to extract the best graph representation of a road scene for solving autonomous scene understanding tasks. By adjusting the modeling strategy to fit the input data distribution jointly with fitting the problem, we can generalize to new data distributions for the same problem. As illustrated in Figure A.1, the rule-based graph extraction method uses fixed rules to define the set of graph edges to extract for each image [38]. As a result, the graphs produced are more static and *domain-specific*. In contrast, our data-driven graph extraction method, RS2G, learns to specialize the set of graph edges to best model the input data, creating more dynamic graph structures that are *domain-adaptive*. Overall, our novel contributions are as follows:

1. We propose RS2G: a data-driven graph extraction and learning methodology for autonomous driving where the structure of the graph and node embeddings are learned dynamically.

2. We prove that RS2G outperforms the state-of-the-art DL-based and GL-based methods at subjective risk assessment. Besides, we analyze RS2G's benefits over rule-based graph extraction via ablation studies.

3. We show that RS2G can better transfer knowledge from simulation to real-world (e.g., Sim2Real) driving than state-of-the-art.

4. We illustrate how RS2G can model several rule-based relations simultaneously with each learned relation and how RS2G enables graph sparsity tuning with minimal performance impacts.

5. We open-source our implementations to the community to foster the development of data-driven graph extraction and modeling approaches at `https://github.com/`

AICPS/RS2G.

As for the remainder of this chapter, Section A.3 discusses related works, and Section A.4 elaborates on our problem formulation and methodology. Section A.5 presents our experimental results, Section A.6 discusses our key findings and analysis, and finally, Section A.7 presents our conclusions.

A.3 Related Works

In this section, we discuss related works on interaction modeling via rule-based graphs and methods for learning data-driven graph representations of visual contexts. We also introduce background on transfer learning in AVs and discuss existing approaches to train generalizable models for effective knowledge transfer.

A.3.1 Interaction Modeling for Autonomous Driving

Recently, several works have found that explicitly modeling interactions between agents in dynamic environments can improve an autonomous system's ability to understand and reason about its environment. Specifically, in the AV use case, multiple works have proposed using domain-knowledge-derived rules to extract graph representations of driving scenarios, denoted as *scene-graphs*. These approaches typically use (i) a perception algorithm to identify the set of agents in the scene and their attributes, (ii) a set of graph-extraction rules to build the graph edges, and (iii) a deep-learning-based graph model, such as the popular multi-relational graph convolution network (MRGCN). [74] proposes a rule-based graph extraction method that encodes relationships such as *Same-lane*, *Following*, *Approaching*, *Overtaking* between road users. They show that their representation enables a graph autoencoder to learn to infer relationships between road users in new scenarios. [164] uses rule-based scene-graph extraction and MRGCN modeling to enable explainable predictions of future driver actions. [38] demonstrates how a rule-based scene-graph improves risk assessment performance over conventional CNN-based methods. [165] uses a rule-based graph extracted from multi-modal sensor data to perform accurate driver action prediction. The aforementioned methods have the same limitation: they rely on rule-based graph extraction. This constraint restricts these approaches to their specialized tasks and data domains, with different tasks requiring new rules to be defined; each work above uses a different set of rules. In our data-driven extraction approach, such overhead is eliminated as we learn the graph extraction rules directly from the data, enabling high performance across tasks and data domains.

A.3.2 Learning Graph Representations for Autonomous Vehicles

Several methods for data-driven graph extraction have been proposed for solving various semantic understanding and reasoning tasks. Graph R-CNN [87] proposes a general framework for scene graph extraction from image data. They use a relation proposal network and an attentional GCN to extract the relation types and prune the graph. This method produces graphs more tailored to visual question-answering tasks (e.g., what is the semantic relationship between two objects in a visual scene?) since the model's primary objective is to extract a domain-specific scene-graph composed of semantically meaningful edges. As a result, the generated graphs are less effective at other modeling tasks and transfer learning. To address the generalization issue, Universal-RCNN [166] uses a transferable Graph R-CNN to propagate semantic information across different domains and improve the transfer learning capabilities of object detectors, however, this transfer learning approach depends on knowledge of the data distribution of both the source and target domains.

Specific to autonomy, [65] proposes a method for extracting graph representations of visual road scenes for driver behavior recognition. This method extracts the node features using object detection models. It extracts the graph adjacency matrix from the processed node features, enabling the model to capture spatial features and learn inter-object relations. However, this approach is limited as it only implements one type of relation and only uses visual features to determine whether or not to add an edge between two nodes.

[167] proposes the graph transformer network, which learns to convert heterogeneous graph-structured data into homogeneous meta-graphs, enabling performance improvements with traditional homogeneous graph models such as GCNs and GATs for node classification. However, by reducing heterogeneous graph data to a single relation type (meta-relation), some information encoded in the edge types is lost. Instead, models that can explicitly handle heterogeneous multigraphs, such as MRGCNs, can better model these structures and increase performance significantly.

A.3.3 Transfer Learning for Autonomous Driving

A key challenge for autonomous driving approaches is generalizing a trained model to unseen real-world scenarios with varying degrees of complexity without performance degradation. Autonomous driving models are typically trained on a large synthetic dataset because developers can easily simulate many traffic conditions, road types, and driving scenarios. *Sim2Real* is a term that describes the capability of a robotic system to effectively transfer knowledge gained from simulation environments to real-world applications [15]. Thus, Sim2Real is a textbook transfer learning problem since the goal of the model is to transfer the knowledge gained from the known source domain (training set) to the unknown test domain (real-world driving) for a specific driving task[168].

Two common forms of transfer learning are *inductive* transfer learning and *transductive* transfer learning [168]. Inductive transfer learning involves learning a general set of rules from the source domain (e.g., training a supervised learning model) before applying them to the test domain. In contrast, transductive transfer learning utilizes some knowledge of the test domain distribution so that characteristics of the source and test domain distributions can be used to adapt the model. This work focuses on the inductive case, which more closely aligns with typical autonomous system use cases (e.g., training ML models using processed/simulated data and testing them in diverse real-world settings). In [169], inductive transfer learning between a CNN-based motion prediction model trained on pedestrian/vehicle trajectories to a model trained on cyclist trajectories is evaluated. The authors find that transferring knowledge from pedestrian motion predictors improves the performance of the cyclist motion predictor. In [170], authors propose transferring knowledge from semi-supervised models using contrastive learning and teacher-student methods to improve trajectory prediction performance. [171] evaluates transfer learning from traditional camera models to event camera models for steering angle prediction. Similarly, [172] transfers spatio-temporal features and uses salient data augmentation to improve sim-to-real transfer performance. Performance improvements are shown for steering angle prediction and collision detection. More recently, [38] demonstrated that graph-based scene modeling improves Sim2Real transfer performance compared to CNN-based methods. Still, this approach remains constrained by the domain-specific rules used for graph extraction, as illustrated in Figure A.1. This limitation can impact the model's ability to adapt to different domains.

A.4 RS2G: Scene-Graph Extraction and Embedding Methodology

In this section, we formulate the problem we aim to solve with data-driven graph extraction and scenario modeling. We target subjective risk assessment as it is a task that necessitates robust scenario understanding and accurate modeling of interactions between agents. Additionally, it is closely related to the important area of ADAS and can directly benefit tasks such as driver hand-off, collision avoidance, and emergency braking. After introducing the problem, we present our methodology for data-driven scene-graph extraction and spatio-temporal graph modeling.

A.4.1 Problem Formulation

The problem of subjective risk assessment can be modeled as inspired by [38]. First, a sequence of sensor data is pre-processed by an object detection model. Next, the outputs are converted to a set of scene graphs. Finally, the scene graphs are converted to a spatio-temporal embedding for performing the subjective risk assessment. At a high level, given that the input to the model I is a sequence of sensor data (e.g., camera images) of length T, and the output of the model is Y is used by the AV, the overall system can be modeled at a high level as:

$$\begin{aligned} & Y = \phi(I); \quad X = \{i_1, i_2, ..., i_T\} \\ s.t. \quad & Y = \begin{cases} 0, & \text{if the driving sequence is safe} \\ 1, & \text{if the driving sequence is risky} \end{cases} \end{aligned} \tag{A.1}$$

where Y represents the subjective risk of the driving scene and ϕ represents the function that maps the inputs I to Y. We can use a machine learning model to approximate the function defined by ϕ, and denote the inference output by the machine learning model as $\hat{Y}$.

Though some machine learning methods, namely CNNs, directly operate on sensor inputs to produce output classifications $\hat{Y}$, these approaches only model pixel-level features and fail to model inter-object relationships for high-level objectives as discussed in Section A.2. However, CNN-based models are good at low-level tasks, such as object detection, since these tasks are less dependent on inter-object semantic relationships. Thus, we first use a pre-trained CNN-based object detection model Ω to extract the set of objects O_t and their attributes D_t from each image $i_t \in I$. Then, these outputs are passed to our graph extraction model Ψ to produce a scene-graph G_t.

$$O_t, D_t = \Omega(i_t) \tag{A.2}$$

$$G_t = \Psi(O_t, D_t) \tag{A.3}$$

Using Ω to extract objects from the scene enables us to efficiently produce scene-graphs from sensor data since we can focus our graph extraction model Ψ on semantic relationship modeling between the extracted objects instead of low-level perception. Once we have extracted all the scene-graphs for the current scene, we pass the collection of graphs G to our spatio-temporal graph embedding model $\hat{\phi}$ to make a risk classification $\hat{Y}$ for the driving scene. Thus, our complete system model can be represented as follows.

$$\hat{Y} = \hat{\phi}(G) \quad s.t. \quad G = \{\Psi(\Omega(i_t)) \, \forall t \in \{1, 2, ..., T\}\} \tag{A.4}$$

We elaborate on the implementation of Ψ, and $\hat{\phi}$ in Sections A.4.2 and A.4.3, respectively.

A.4.2 Data-Driven Scene-Graph Extraction

Algorithm 5 presents our approach for extracting a scene-graph G from a set of objects O and their attributes D. We model our scene-graphs as directed, heterogeneous multi-graphs. In this model, objects in the scene are modeled as nodes, and edges model relationships between objects. Each node contains an attribute vector, each edge is a directed edge with binary types, and multiple types of edges can exist between any two nodes. In contrast to the state-of-the-art method from [38], which uses fixed domain-knowledge rules to define the conditions for building each graph edge (e.g., threshold-based distance relations, compass-based directional relations), RS2G uses a data-driven edge encoder Enc_{edge} to generate *domain-specialized* edge types and learns the rules for building each edge *dynamically*.

Our approach consists of the following steps. First, a node encoder model Enc_{node} converts the attributes d_j of each object o_j into a set of encoded node features h_j. Next, we take each pair of nodes, concatenate their feature vectors, and pass the resulting vector to an edge encoder Enc_{edge}. The edge encoder aims to infer if there is an edge of a given relation type r between node j and node k, given their features h_j and h_k, respectively. Each relation type has a different set of learnable weights in the edge encoder, enabling it to learn different rules for constructing each relation type. After the edge encoder has processed all node pairs, the result is an $n \times n \times R$ adjacency matrix for n total nodes and R relation types. This adjacency matrix A and the node features H form the scene-graph G.

Algorithm 5: Data-driven scene graph extraction

```
Input: Objects $O$ and their attributes $D$ at time $t$.
Output: Scene-graph $G$.
def $\Psi(O, D)$:
    $H \leftarrow \{\ \}$
    $A \leftarrow \{\{\{\ \}\}\}$                          // adjacency matrix
    for $o_j, d_j \in O, D$ do
        $h_j \leftarrow Enc_{node}(o_j, d_j)$                   // node encoding
    end
    $combos \leftarrow {}_H C_2$                  // get all node combinations
    for $r \in R$ do
        for $h_j, h_k \in combos$ do
            /* get edges for relation $r$                              */
            $A_{r,j,k} \leftarrow Enc_{edge}(r, h_j, h_k)$
        end
    end
    $G \leftarrow \{H, A\}$                        // add nodes & edges to G
    return $G$
```

A.4.3 Spatio-Temporal Graph Embedding Model

We use a combination of graph modeling and sequence modeling components to spatio-temporally model a sequence of scene-graphs for risk assessment. Open-source code for our model is provided at `https://github.com/AICPS/RS2G`. Our model architecture is illustrated in Figure A.2 and consists of the following steps. First, an object detection model extracts the set of objects in the scene, which are then processed into a graph using Algorithm 5. The resulting graphs are passed to our spatial graph model to produce a set of graph embeddings. These graph embeddings are then temporally modeled to produce a final risk assessment for the driving scene. The spatial graph embedding model we use is the same as the one presented in Chapter 2. The temporal model is derived from [38] for the task of risk assessment and is detailed below. Our complete risk-assessment workflow is detailed in Section A.4.4.

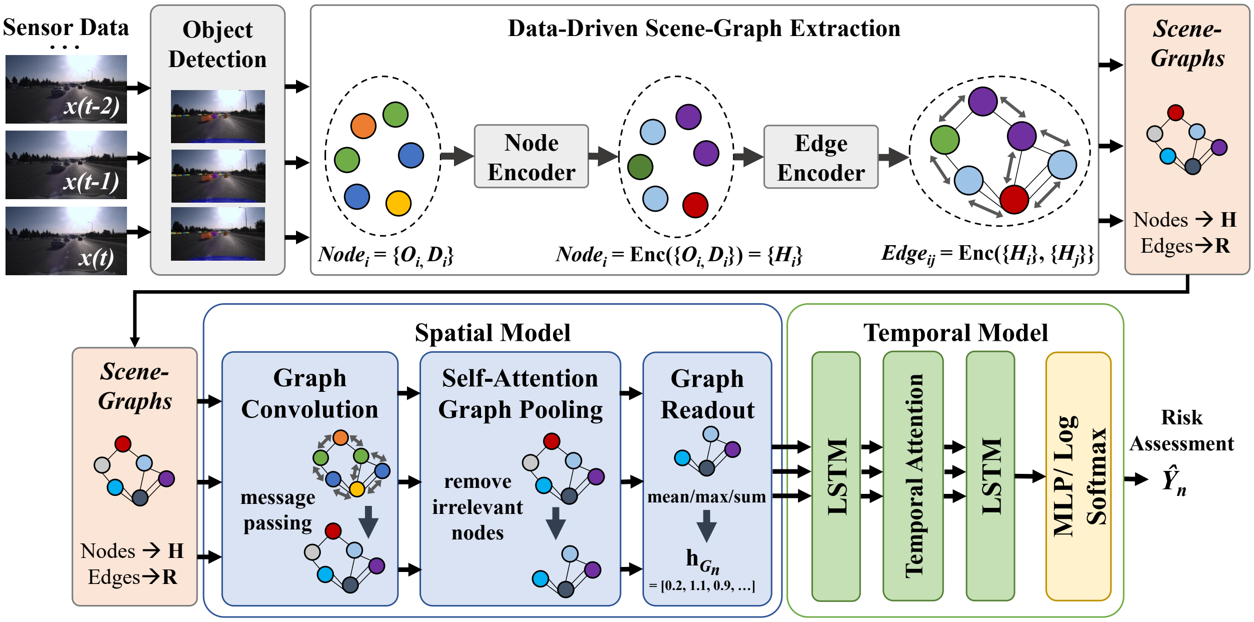


Figure A.2: The architecture of RS2G. First, an object detector extracts the set of objects from the input sensor data. Then, our data-driven scene-graph extraction approach generates node embeddings and heterogeneous graph edges. Then, these scene-graphs are spatio-temporally modeled using an MRGCN and an LSTM. The output embedding generated by the LSTM is processed by an MLP to infer if the scenario is *risky* or *non-risky*.

Temporal Modeling

After the set of scene-graphs is spatially modeled by the MRGCN, the sequence of scene-graph embeddings is temporally modeled by a long short-term memory (LSTM) [173] network. For each timestamp t, the LSTM updates the hidden state p_t and cell state c_t as follows,

$$p_t, c_t = \mathbf{LSTM}(\mathbf{h}_{G_t}, c_{t-1}), \tag{A.5}$$

where $\mathbf{h}_{G_t}$ is the final *scene-graph* embedding from timestamp t. After the LSTM processes all the scene-graph embeddings, a temporal readout operation is applied to the resultant output sequence to compute the final spatio-temporal embedding Z given by

$$\mathbf{Z} = \mathbf{TEMPORAL\_READOUT}(p_1, p_2, ..., p_T) \tag{A.6}$$

where the **TEMPORAL_READOUT** operation could be the extraction of only the last hidden state p_T (LSTM-last) or could be a temporal attention layer (LSTM-attn).

In [38], adding an attention layer b between successive LSTM layers improves performance in AV risk assessment and consists of the following components. *LSTM-attn* calculates a context vector q using the hidden state sequence $\{p_1, p_2, ..., p_T\}$ returned from the LSTM encoder layer as given by

$$q = \sum_{t=1}^{T} \beta_t p_t \tag{A.7}$$

where the probability β_t reflects the importance of p_t in generating q. The probability β_t is computed by a *Softmax* output of an energy function vector e, whose component e_t is the energy corresponding to p_t. Thus, the probability β_t is formally given by

$$\beta_t = \frac{\exp(e_t)}{\sum_{k=1}^{T} \exp(e_k)}, \tag{A.8}$$

where the energy e_t associated with p_t is given by $e_t = b(s_0, p_t)$. The temporal attention layer b scores the importance of the hidden state p_t to the model objective (e.g., binary classification for risk assessment). The variable s_0 in the temporal attention layer b is computed from the last hidden representation p_T. The input sequence's final spatio-temporal embedding, Z, is produced by feeding the context vector q to an LSTM decoder layer.

Output Classification

The last layer in our model generates an output risk classification $\hat{Y}$ from the spatio-temporal embedding Z as follows.

$$\hat{Y} = Softmax(\mathbf{MLP}(Z)) \tag{A.9}$$

Our model is implemented as a binary classifier, so we use Cross-Entropy Loss to train the model. Next, we discuss the end-to-end workflow of our approach.

A.4.4 Risk Assessment Workflow

Algorithm 6 defines our complete workflow for risk assessment from sensor data. First, each sensor input in I is processed by an object detector Ω to get the set of visible objects in the scene O_t and their attributes D_t. This result is processed by the graph extraction model Ψ to produce a scene-graph G_t for each timestep t in I. The sequence of extracted graphs is then passed to the risk assessment model $\hat{\phi}$, which uses its spatial model to convert each scene-graph into a graph embedding. These embeddings are then temporally modeled to produce a spatio-temporal sequence embedding Z. Finally, Z passed through an MLP to classify the risk of the driving scenario as risky (1) or non-risky (0).

Algorithm 6: Complete workflow for scene-graph extraction and risk assessment

```
Input: Sequence of sensor inputs $I$.
Output: Risk assessment $\hat{Y}$.
def $\hat{\phi}(G)$:
    $h_G \leftarrow \{ \}$
    for $G_t$ in $G$ do
        $\mathbf{h}_{G_t} \leftarrow$ SPATIAL_MODEL($G_t$)
    end
    $Z \leftarrow$ TEMPORAL_MODEL($\mathbf{h}_G$)
    $\hat{y}_0, \hat{y}_1 \leftarrow Softmax(\text{MLP}(Z))$
    if $\hat{y}_1 \geq \hat{y}_0$ then
        return 1                                   // risky
    else if $\hat{y}_0 > \hat{y}_1$ then
        return 0                                   // not risky
def ASSESS_RISK($I$):
    $G \leftarrow \{ \}$
    for $i_t$ in $I$ do
        $O_t, D_t \leftarrow \Omega(i_t)$                 // detect objects
        $G_t \leftarrow \Psi(O_t, D_t)$                   // extract graphs
    end
    $\hat{Y} \leftarrow \hat{\phi}(G)$                     // predict risk
    return $\hat{Y}$
$\hat{Y} \leftarrow$ ASSESS_RISK($I$)
```

A.5 Experiments

In this section, we present our experiments comparing our data-driven scene-graph extraction and learning methodology with current state-of-the-art graph-based and DL-based approaches. We first discuss our experimental setup, training procedure, and key metrics. Next, we present results for subjective risk assessment across diverse driving datasets. We also evaluate the Sim2Real transfer learning capability of each method and perform an ablation study demonstrating the benefits of our data-driven graph extraction method. Finally, we analyze the key differences between the relations and graph structures learned by RS2G and the rule-based relations used by state-of-the-art graph extraction methods.

A.5.1 Experimental Setup

We used three different types of datasets for our experiments: (i) simulated lane change scenarios of varying risk in the CARLA driving simulator [63]; (ii) real-world, clear-weather safe driving in the California Bay Area from the Honda Driving Dataset [94]; and (iii) real-world crashes and dangerous road scenarios from dash-cam footage in the Detection of Traffic Anomaly Dataset [95]. We refer to these datasets as (i) *271-syn* and *1043-syn*, (ii) *1361-honda*, and (iii) *620-dash*, respectively, with the number indicating the number of driving clips in each dataset. Each driving clip is between 10-40 seconds in duration. We also used a subset of *1361-honda* consisting of only lane-changing clips, denoted as *571-honda* for our transfer learning experiments. For further details about dataset preparation, please refer to Chapter 2. We use the roadscene2vec library [93] to represent the scene-graphs, train the models, and perform the evaluation in PyTorch.

Our proposed model comprises three main modules, graph extraction, spatial model, and temporal model. For graph extraction, we implement two variants with RS2G: 1D MLP and 2D MLP, corresponding to the number of MLP layers used for the encoders (1 layer and 2 layers). The 1D MLP has a node encoder shape of 15x15 and an edge encoder shape of 30x15, since the edge encoder takes in the features of two nodes at a time. The 2D MLP has a node encoder shape of 15x15x15 and an edge encoder shape of 30x30x15. Our spatial and temporal modeling components follow the structure of the MRGCN+LSTM model from [38] for fair comparison. Thus, for modeling spatial graph features, we use a 2-layer MRGCN with Self-Attention Graph Pooling and *mean* readout. We use a 2-layer LSTM with temporal attention as the temporal readout operation. Since we model risk assessment as a binary classification task, we evaluate each model in terms of Accuracy, Matthews Correlation Coefficient (MCC) [97], and Area Under the ROC Curve (AUC) [96]. Accuracy in this case is the standard metric indicating the percentage of correctly classified scenes. AUC score is a typical metric for scoring classifiers across multiple decision boundaries; it ranges from 0.0 to 1.0 with higher performance indicating a more robust model. MCC score is considered a more reliable metric than accuracy for scoring models on imbalanced datasets; an MCC score of -1.0 represents an always wrong classifier, 1.0 represents an always correct classifier, and 0.0 represents a random classifier. The number of risky and non-risky scenes in each dataset is shown in Table A.1.

Dataset	Non-Risky Scenes	Risky Scenes	Non-Risky:Risky Ratio
271-syn	223	48	4.65:1
571-honda	475	99	4.80:1
620-dash	323	297	1.09:1
1043-syn	898	146	6.15:1
1361-honda	1207	154	7.84:1

Table A.1: Number of risky and non-risky scenarios in each dataset. The ratio indicates the label distribution between negative and positive samples.

Regarding baseline models, we primarily compare with the rule-based graph-extraction and learning approach proposed in [38] as it is a state-of-the-art method for this task. We denote this method as ”Rule-Based” graph extraction in the experiments. We also compare against the previous state-of-the-art: the CNN+LSTM architecture from [58]. Since this method does not use graphs, we denote its graph extraction as ”None.”

For training and evaluating each model, we use a 7-to-3 train-test split for each dataset. The only exception is the transfer experiments, where we train with 70% of the training dataset and evaluate with 100% of the testing dataset since they are distinct. We used a Linux server with an Intel Xeon E5 CPU and an Nvidia Titan Xp GPU for training and evaluating each model. Notably, the RS2G models were slower to train since they needed to train the graph encoder layers in addition to the MRGCN and LSTM layers. Additionally, the edge density of the graphs affected training time since each edge adds graph convolution operations, enabling the sparser rule-based graphs to train faster. Training speed and convergence could be improved in future work if graph sparsity is encouraged during RS2G training (e.g., via additional training objectives).

A.5.2 Experiment I: Risk Assessment Performance

Table A.2 shows the performance of each model variant at risk assessment on the datasets. We present results for the synthetic datasets (*271-syn, 1043-syn*) as well as the real-world driving datasets (*620-dash, 1361-honda*). Overall, all the models demonstrate notably lower learning quality (i.e., accuracy, MCC, and AUC) for real-world driving datasets than synthetic datasets. However, models utilizing 1D MLP and 2D MLP graph extraction techniques suffer from less degradation and are more able to provide effective performance. In particular, the CNN+LSTM with no graph extraction shows a weak MCC score across all datasets (i.e., only slightly better than a random classifier), indicating it cannot distinguish positive and negative instances well.

Across all datasets, using 1D MLP and 2D MLP graph extraction layers provide significantly higher accuracy than those using "None" (CNN+LSTM) or rule-based graph extraction. On average, the 2D MLP graph extraction technique provides 25.99% higher accuracy than the CNN+LSTM and 4.29% higher accuracy than rule-based graph extraction. Additionally, our 1D MLP and 2D MLP provide higher MCC and AUC scores, indicating that using 1D MLP and 2D MLP graph extraction techniques has a decisive advantage in distinguishing positive and negative samples. In contrast, using none graph extraction with CNN+LSTM provides significantly lower learning quality. In particular, for a real-world imbalanced dataset with more crashes and risky scenarios (*620-dash*), CNN+LSTM delivers seriously degraded accuracy and an MCC score worse than a random classifier (less than 0.0). 2D MLP graph extraction also has reduced performance for *620-dash*, but it is still significantly better than the baselines.

Dataset	Model		Acc.	MCC	AUC
	Graph Ext.	Downstream			
271-syn	None [58]	CNN+LSTM	73.17%	0.1887	0.8043
	Rule-Based [38]	MRGCN+LSTM	87.80%	0.6140	0.9676
	RS2G (1D MLP)	MRGCN+LSTM	90.24%	0.6583	0.9643
	RS2G (2D MLP)	MRGCN+LSTM	**93.90%**	**0.7876**	**0.9850**
1043-syn	None [58]	CNN+LSTM	71.66%	0.1111	0.7173
	Rule-Based [38]	MRGCN+LSTM	95.86%	0.8238	0.9861
	RS2G (1D MLP)	MRGCN+LSTM	97.77%	0.9055	0.9888
	RS2G (2D MLP)	MRGCN+LSTM	**97.77%**	**0.9055**	**0.9898**
1361-honda	None [58]	CNN+LSTM	60.39%	0.0391	0.7110
	Rule-Based [38]	MRGCN+LSTM	86.31%	0.2445	0.9341
	RS2G (1D MLP)	MRGCN+LSTM	87.04%	0.1626	0.9315
	RS2G (2D MLP)	MRGCN+LSTM	**89.00%**	**0.3029**	**0.9383**
620-dash	None [58]	CNN+LSTM	48.92%	-0.1749	0.5256
	Rule-Based [38]	MRGCN+LSTM	70.97%	0.4230	0.7804
	RS2G (1D MLP)	MRGCN+LSTM	68.82%	0.3967	0.7403
	RS2G (2D MLP)	MRGCN+LSTM	**77.42%**	**0.5620**	**0.8358**

Table A.2: Risk Assessment Performance for different graph extraction methods and datasets. Rule-Based graph extraction is derived from [38], while RS2G is our proposed approach. "Downstream" indicates the type of model processing the graph representation, with CNN+LSTM using raw images and MRGCN+LSTM using scene-graphs as input.

A.5.3 Experiment II: Transfer Learning Evaluation

Here we evaluate the Sim2Real transfer learning capability of each model. To perform the evaluation, we first train each model on one of the simulation datasets (*271-syn* or *1043-syn*), then evaluate the trained model on a real-world dataset (*620-dash*). The analysis is two-fold: (i) the driving behaviors differ from the simulation (lane changes only) to the real-world datasets (all driving maneuvers are present), and (ii) the visual context differs between the simulation environment and the real-world scenarios. The results of this experiment are shown in Table A.3. In particular, we train our models on simulated datasets *271-syn* and *1043-syn* and perform transfer learning on *620-dash*, where there are many instances

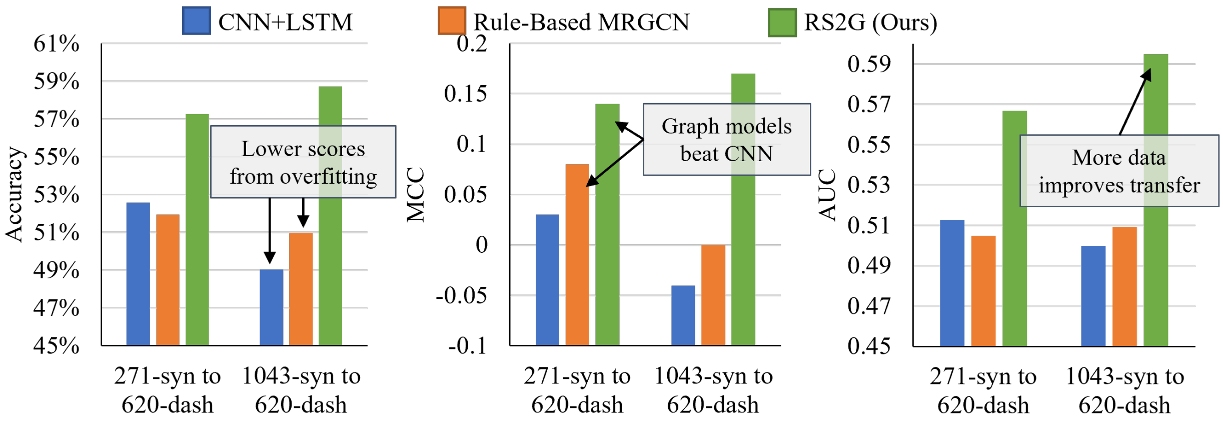


Figure A.3: Transfer learning comparison between different graph extraction methods: None (CNN+LSTM model) [58], Rule-Based (MRGCN+LSTM model) [38], and RS2G with 2D MLP (MRGCN+LSTM model). Results are denoted as "(Train Dataset) to (Test Dataset)".

of crash scenarios. As shown, RS2G achieves notably higher accuracy, MCC, and AUC than the CNN+LSTM model and rule-based graph extraction with the MRGCN. Since 620-dash includes significantly more risky scenarios and differs greatly from the simulated datasets, all learning methods demonstrate degraded performance. However, RS2G suffers from notably less degradation and shows higher accuracy, MCC, and AUC after transfer than the other methods. Notably, the two baselines (CNN+LSTM and Rule-Based MRGCN) trained on *1043-syn* perform worse after transfer than the same models trained on *271-syn*, likely because these models overfit to their training domain (synthetic data) at a detriment to the test domain performance (real-world driving). In contrast, RS2G can generalize well across dataset sizes without overfitting in the same manner, resulting in higher transfer performance when trained on *1043-syn*. We note that, when evaluated on datasets that are more closely related to the training domain (e.g., lane change scenes in the *571-honda* dataset), RS2G can achieve higher performance figures (82% accuracy), likely because the data distribution and action space are closer to the training data.

A.5.4 Experiment III: Ablation Study

To demonstrate the contributions of each sub-component of our proposed methodology, we present an ablation study in Table A.3. We evaluate rule-based graph extraction vs. different variants of our data-driven graph extraction methodology. As demonstrated in Table A.3, both 1D MLP and 2D MLP graph extraction techniques provide better accuracy, MCC, and AUC than rule-based graph extraction. Using MRGCN for the spatial model delivers better accuracy, MCC, and AUC than using MLP in rule-based and 1D MLP graph extraction, demonstrating the benefits of explicitly modeling the relations between agents. Additionally, using LSTM for the temporal model provides better accuracy than using mean, likely because the LSTM can better model temporal patterns in the graph embeddings. Notably, the 1D MLP graph extractor with the MLP+LSTM model outperforms the rule-based graph extractor with the MRGCN+LSTM model, indicating that our data-driven graphs produce a higher quality representation that is easier for multiple kinds of downstream models to classify, not just MRGCNs. Comparing models using 1D MLP and 2D MLP graph extractors, the model using a 2D MLP layer has slightly lower accuracy while demonstrating stronger MCC and AUC scores, indicating a slightly better capability and distinguishing between positive and negative samples.

Graph Ext.	Spatial Model	Temporal Model	Acc.	MCC	AUC
Rule-Based [38]	MLP	mean	52.15%	0.0000	0.4973
Rule-Based [38]	MLP	LSTM	62.90%	0.2741	0.6811
Rule-Based [38]	MRGCN	mean	63.44%	0.2696	0.6867
Rule-Based [38]	MRGCN	LSTM	**75.27%**	**0.5197**	**0.8248**
RS2G (1D MLP)	MLP	mean	61.29%	0.2284	0.6436
RS2G (1D MLP)	MLP	LSTM	72.04%	0.4572	0.8062
RS2G (1D MLP)	MRGCN	mean	68.28%	0.3865	0.7154
RS2G (1D MLP)	MRGCN	LSTM	**78.49%**	**0.5746**	**0.8514**
RS2G (2D MLP)	MRGCN	LSTM	**77.96%**	**0.5784**	**0.8618**

Table A.3: Ablation study across different graph extraction methods, spatial models, and temporal models. Models are trained and evaluated on *620-dash*.

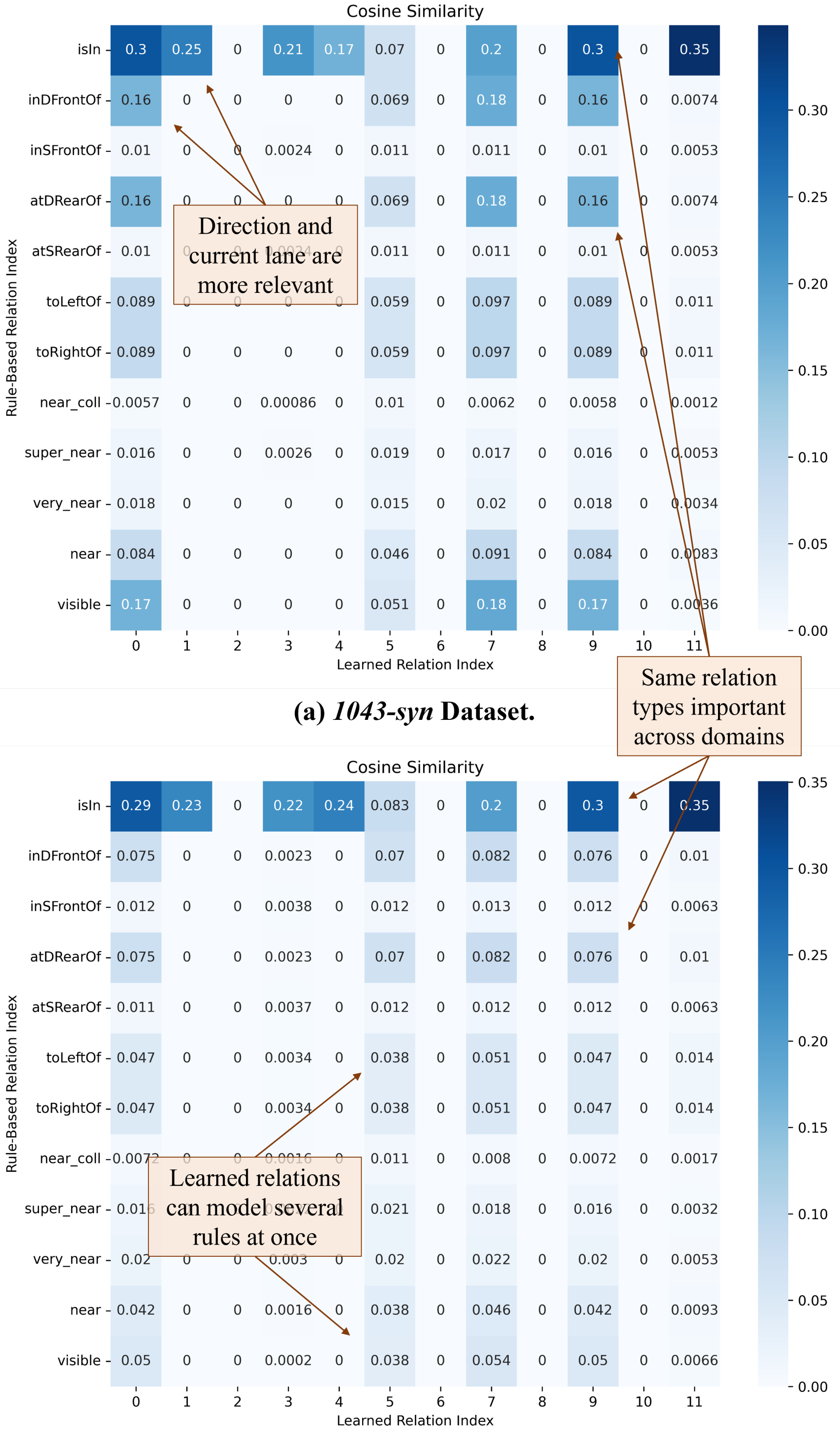
A.5.5 Experiment IV: Learned vs. Rule-Based Relation Analysis

Data-driven graph extraction can result in significantly different graph structures than rule-based graph extraction. To analyze the differences between these two extraction methods,the following experiments evaluate (i) the similarity between the relations learned by RS2G and the rule-based relations from the baseline [38], and (ii) graph structure metrics for each method.

Cosine Relation Similarity

We compare the cosine similarity between the data-driven relations learned by RS2G and the set of rule-based relations extracted by [38] for the *1043-syn* dataset in Figure A.4. Cosine similarity is a popular metric for comparing sparse vectors and is often used to compute document similarity over term-frequency vectors [174]. We use cosine similarity to compare the adjacency matrices extracted by rule-based graph extraction with the data-driven adjacency matrices produced by RS2G to determine if the extracted graphs can implicitly learn some of the rules from the data itself. Our graphs are 3-dimensional binary adjacency matrices, with an NxN matrix for each relation $r \in R$ for a graph with N nodes. For each of our 12 learned relations, we compute the cosine similarity of its adjacency matrices with those for each rule-based relation, averaging across all graphs in the dataset.

The similarity in relations highlighted across datasets supports the fact that our model can effectively transfer knowledge; the relations that the model deems relevant to the task are relevant across datasets. Additionally, the differences in the relative weights of these relations across datasets show how our model specializes to different datasets by adjusting the density of different relation types depending on the data. This finding supports our point that data-driven graph extraction can boost performance while maintaining generalization ability.



(a) *1043-syn* Dataset.

(b) *620-dash* Dataset.

Figure A.4: Cosine similarity between relations learned by RS2G (2D MLP) and the set of rule-based relations used in [38] for the synthetic *1043-syn* dataset and the real-world *620-dash* dataset.

Graph Structure Comparison

In addition to measuring the similarity of different relation types, we also evaluate the structural differences between rule-based graphs and the data-driven graphs extracted by RS2G. We evaluate how the methods differ regarding graph sparsity and edge distribution and correlate these metrics with risk assessment accuracy. We also assess how increasing RS2G's edge extraction threshold (γ) affects the sparsity of generated graphs and the model's overall performance. The threshold γ indicates the sigmoid score that Enc_{edge} must overcome to add a given edge to the graph, so higher γ results in sparser graphs. Our results using 2D MLP graph extraction are shown in Table A.4. 2D MLP extraction with various thresholds exhibits higher accuracy than rule-based graph extraction. We achieved the best accuracy using $\gamma = 0.5$, so this setting was used for the rest of the experiments shown in the chapter. Using $\gamma = 0.25$ lowers the performance, possibly due to overfitting. On the other hand, using $\gamma = 0.75$ also offers lower accuracy since fewer features are extracted.

Graph Ext.	Acc.	Avg. Deg.	Avg. Edges	σ Edges
Rule-Based [38]	95.86%	3.84	16.50	10.51
RS2G (2D MLP, $\gamma = 0.25$)	96.82%	78.58	636.22	556.10
RS2G (2D MLP, $\gamma = 0.5$)	**97.77%**	**36.42**	**286.84**	**231.07**
RS2G (2D MLP, $\gamma = 0.75$)	96.82%	5.92	46.74	38.13

Table A.4: Comparison of graph structure metrics across different extraction methods. γ represents the edge extraction decision threshold. RS2G(2D MLP, $\gamma = 0.5$) has the highest accuracy.

A.6 Discussion

Overall, our experimental results support our claim that *data-driven* graph extraction improves representation quality and generalization over that of *rule-based* graph extraction, resulting in performance improvements on both simulated and real-world datasets. Our results also demonstrate that RS2G can more effectively perform Sim2Real transfer learning, support configurable graph sparsity, and learn relations overlapping with multiple rule-based relations to improve modeling. Next, we discuss how RS2G can be deployed in a real-world autonomous system, the limitations of our research scope, and potential future research directions.

A.6.1 Practicality

Fundamentally, RS2G leverages existing deep-learning and graph-learning libraries for execution, so integration with a typical autonomous computing platform should follow standard training, validation, and model compilation procedures. The complexity of deploying RS2G will be the same as deploying a rule-based graph model since the primary difference is the addition of a few layers for node/edge encoding. The requisite inputs and outputs are also present in most autonomous driving and ADAS pipelines (e.g., camera inputs, object detection model, ADAS control system). Regarding utility, RS2G's output can be sent to the ADAS system to inform tasks such as driver control handoff, emergency braking, and dynamic driving profile adjustment.

A.6.2 Limitations and Future Work

Although RS2G demonstrates improved generalization capabilities compared to rule-based graph extraction, further improvements could be realized by adding auxiliary training objectives such as self-supervision or Kullback-Liebler Divergence (KLD) loss across adjacency matrices of each relation type. By adding a learned self-supervision component between the node and edge encoder inputs and their output node embeddings and adjacency matrices, the node and edge encoders would learn to model an invertible function mapping from the inputs to the outputs and may, as a result, produce a more general graph representation. Similarly, adding a KLD loss would encourage the model to learn different adjacency matrices for each relation type, potentially improving the quality of the encoded representation in the scene-graph and, subsequently, the model's generalization ability. We conducted preliminary experiments to test these theories, but the results were inconclusive. Thus, we leave the further study of these topics for future work.

Another potential source of improvement is the node and edge encoder models. In this work, we exhaustively studied how 1D and 2D MLPs could enable data-driven graph extraction and benefit the model. However, more complex deep-learning approaches could be leveraged to improve expressive quality. Further, rule-based graphs could be integrated with learned graphs (e.x., by combining their adjacency matrices) to leverage the benefits of both methods. However, this strategy may only benefit certain application domains, so we leave this for future work.

In terms of application domains, we studied subjective risk assessment for AVs in this chapter. Still, AVs must perform various tasks to perceive, plan, and maneuver safely. Tasks such as localization, motion prediction, and path planning involve semantic scene understanding, and graph-based modeling approaches have been shown to improve performance at these tasks [163]. Thus, graph-based methods in these applications could benefit from using a learned scene-graph representation similar to RS2G.

A.7 Summary

This chapter presents RS2G, an innovative data-driven graph extraction approach that learns to optimize the graph topology for each domain. We show that RS2G produces better quality representations resulting in higher performance at subjective risk assessment across diverse driving datasets than state-of-the-art methods. RS2G also significantly improves generalization and outperforms state-of-the-art methods at transferring knowledge gained from synthetic datasets to more complex, unpredictable, and risky real-world scenarios. Additionally, our ablation study showed clear benefits from using data-driven graph extraction compared to rule-based graph extraction. RS2G also produced a representation that could be modeled more effectively even by simpler downstream models. Finally, we demonstrated how each relation learned by RS2G could model multiple domain-knowledge-defined rules simultaneously and how the sparsity of graphs can be dynamically tuned while mitigating performance impacts. Our findings open the door for deep exploration into data-driven graph extraction and graph structure optimization in future works. Our results demonstrate the power of data-driven graph extraction and its applicability to various autonomous systems and scenario-understanding applications.

Appendix B

Other Research Areas

B.1 Overview

In addition to the research areas discussed in the main body of this dissertation, I have also studied several other research topics. Below, these topics and their related publications are briefly summarized.

B.2 Machine Learning for Advanced Manufacturing

B.2.1 Acoustic Side-Channel Attacks on DNA Synthesis Machines

Synthetic biology is developing into a promising science and engineering field. One of the enabling technologies for this field is the DNA synthesizer. It allows researchers to custom-build sequences of oligonucleotides (short DNA strands), which are valuable intellectual property. Incorporating these sequences into organisms can result in improved disease resistance and lifespan for plants, animals, and humans. However, these DNA synthesizers are fully automated systems with cyber-domain processes and physical domain components and may be prone to security breaches like any other computing system. In [175], we present a novel acoustic side-channel attack methodology which can be used on DNA synthesizers to breach their confidentiality and steal valuable oligonucleotide sequences. In addition, we reconstruct DNA sequences to show how effectively an attacker with biomedical-domain knowledge would be able to derive the intended functionality of the sequence using the proposed attack methodology. More details are presented in [176, 177].

B.2.2 Sabotage Attack Detection in Additive Manufacturing

AM, or 3D Printing, is seeing practical use for the rapid prototyping and production of industrial parts. The digitization of such systems not only makes AM a crucial technology in Industry 4.0 but also presents a broad attack surface that is vulnerable to sabotage attacks [178]. In the field of AM security, sabotage attacks are cyberattacks that introduce inconspicuous defects to a manufactured component at any specific process of the AM digital process chain, resulting in the compromise of the component's structural integrity and load-bearing capabilities [179]. Defense mechanisms that detect such attacks using side-channel analysis have been studied [180, 181, 179, 182]. However, most current works focus on modeling the state of AM systems using a single side-channel, thus limiting their effectiveness at attack detection. To address this challenge, we present a novel multi-modal sabotage attack detection system for Additive Manufacturing (AM) machines in [183]. By utilizing multiple side-channels (*e.g.*, vibration, acoustic, magnetic, power), we improve system state estimation significantly over that of existing methods. Besides, we analyze the value of each side-channel for performing attack detection in terms of mutual information shared with the machine control parameters. More details on how side-channels can be leveraged in manufacturing systems are provided in [184].

B.2.3 Neuromorphic Computing for the Predictive Maintenance of Manufacturing Systems

If machine failures can be detected preemptively, then maintenance and repairs can be performed more efficiently, reducing production costs. Many machine learning techniques for performing early failure detection using vibration data have been proposed; however, these methods are often power and data-hungry, susceptible to noise, and require large amounts of data preprocessing. Also, training is usually only performed once before inference, so they do not learn and adapt as the machine ages. Thus, [185] proposes a method of performing online, real-time anomaly detection for predictive maintenance using Hierarchical Temporal Memory (HTM). Inspired by the human neocortex, HTMs learn and adapt continuously and are robust to noise. Our approach outperforms both state-of-the-art deep learning and statistical algorithms at preemptively detecting real-world cases of bearing failures and simulated 3D printer failures.

B.3 Automotive Security

B.3.1 In-Vehicle Network Security

The complexity of automotive E/E systems is increasing with time as discussed in Chapter 1. Modern systems use highly interconnected hardware and software systems, presenting a broad attack surface for adversaries. Specifically, the in-vehicle network stack is closely tied to critical functionality such as steering, braking, acceleration, all of which are controlled electronically in modern vehicles. Further, systems such as door locks and alarms, active safety systems, airbags, and infotainment are all connected via shared buses and interconnects. This fact enables adversaries to exploit a vulnerability in one subsystem to attack other critical subsystems. Furthermore, the emergence of Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication protocols presents an even broader attack surface. In [186] we explore the taxonomy of in-vehicle and V2V/V2I networks, security vulnerabilities of these networks, and current methods for exploiting these vulnerabilities. Overall, modern vehicles are highly exposed to attacks, and the potential impact of attackers will increase with the development of autonomous driving systems, V2X communication, and other advanced technologies. More details on the security risks associated with connected vehicles are discussed in [187].

B.3.2 Physical Layer Key Generation for Secure V2X Communication

An emerging paradigm for intelligent transportation is the integration of communication capabilities between vehicles and infrastructure, collectively denoted as vehicle-to-everything (V2X) communication. V2X promises to improve road safety, reduce congestion, and reduce the complexity of designing autonomous vehicles by allowing vehicles and signaling equipment to broadcast their intentions. However, due to the sensitivity of the information exchanged in Vehicle-to-Everything (V2X) communication, generating secret keys is critical to secure these communications to prevent spoofing and false data injection [186]. As nature is open access, distributed symmetric keys are more vulnerable to attacks in the vehicular environment. Additionally, V2X applications necessitate low-latency communication for rapid decision making, so asymmetric encryption is less practical. Physical layer key generation methods using wireless channel characteristics show promise in preventing such attacks by enabling the keys to be generated independently by each party and removing the need for distribution [188, 189]. In [190], we present a novel key generation approach in a real vehicular environment based on Channel State Information (CSI), including a new algorithm for key bit extraction. We implement our algorithm using USRP B210 Software-Defined Radios (SDR) and the industry-standard V2X communication protocol: IEEE 802.11p. The proposed key generation protocol uses the CSI values of each sub-carrier as a source of randomness. We compare our technique to state-of-the-art Received Signal Strength (RSS)-based approaches, and show that our method achieves better performance.

B.4 Autonomous Vehicle Motion Prediction

B.4.1 Context-Aware Dynamic Architectures for Robust Motion Prediction

The ability of an autonomous vehicle to accurately predict the motion of other road users across a wide range of diverse scenarios is critical for both motion planning and safety. In modular AV architectures, the path planning and control modules depend on the outputs of the motion prediction module to understand where other road users are likely to be in the future. However, existing motion prediction methods do not explicitly model contextual information about the environment which can cause significant variations in performance across diverse driving scenarios. To address this limitation, we propose **CASTNet**: a dynamic, context-aware approach for motion prediction that (i) identifies the current driving context using a spatio-temporal model, (ii) adapts an ensemble of motion prediction models to fit the current context, and (iii) applies novel trajectory fusion methods to combine predictions output by the ensemble. This approach enables CASTNet to improve robustness by minimizing motion prediction error across a wide range of scenario types. We demonstrate how CASTNet can improve both CNN-based and graph-learning-based motion prediction approaches and conduct ablation studies on the performance, latency, and model size for various architecture choices.