Title
Architecture and System Support for Safety-aware Autonomous Vehicle Design

Author
Zhao, Hengyu

Publication Date
2021

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA SAN DIEGO

Architecture and System Support for Safety-aware Autonomous Vehicle Design

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science (Computer Engineering)

by

Hengyu Zhao

Committee in charge:

Professor Jishen Zhao, Chair
Professor Henrik I. Christensen
Professor Sujit Dey
Professor Ryan Kastner
Professor Steven Swanson

2021

The Dissertation of Hengyu Zhao is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2021

DEDICATION

To my family.

EPIGRAPH

If you can't explain it simply, you don't understand it well enough.

Albert Einstein

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my advisor Professor Jishen Zhao for her guidance, support and encouragement during my Ph.D. career. By working with her, I learned how to do research, and write a research paper independently. Under her guidance, I started to explore unsolved but exciting topics in computer architecture and autonomous driving. I believe the experience will be invaluable for me and my future careers.

I would like to thank my dissertation committee members: Professor Henrik I. Christensen, Professor Sujit Dey, Professor Ryan Kastner and Professor Steven Swanson for their valuable comments and feedback.

I would like to acknowledgement my collaborators, Siva Kumar Sastry Hari, Timothy Tsai, Michael B. Sullivan, Stephen W. Keckler, Pingfan Meng, Yubo Zhang, Michael Wu, Haolan Liu, Tiancheng Lou, Hui Shi, Li Erran Li, Sabur Baidya, Yu-Jen Ku, Sujit Dey, Saransh Gupta, Mohsen Imani, Fan Wu, Tajana Rosing, Jiawen Liu, Matheus A. Ogleari, Dong Li, Linuo Xue, Ping Chi, Colin Weinshenker, Mohamed Ibrahim, Adwait Jog, for their collaboration and help. This work cannot be finished without their efforts. I would like to thank my labmates and I am grateful to have worked with them.

Most importantly, I would like to thank my family, for their patience, understanding and love. They always support and encourage me to overcome any difficulty and challenge throughout my life.

Chapter 2 contains material from "Processing-in-Memory for Energy-efficient Neural Network Training: A Heterogeneous Approach" by Hengyu Zhao, Jiawen Liu, Matheus A. Ogleari, Dong Li, Jishen Zhao, which appears in International Symposium on Microarchitecture (Micro), Oct. 2018. The dissertation author was the primary investigator and author of this material.

Chapter 3 contains material from "Driving Scenario Perception-Aware Computing System Design in Autonomous Vehicles", which appears in IEEE International Conference on Computer Design (ICCD), Oct, 2020, "Safety Score: A Quantitative Approach to Guiding Safety-Aware

Autonomous Vehicle Computing System Design", which appears in IEEE Intelligent Vehicles Symposium, Oct, 2020 and "Towards Safety-Aware Computing System Design in Autonomous Vehicles", which appears in arXiv repo. All these three papers are worked by Hengyu Zhao, Yubo Zhang, Pingfan Meng, Hui Shi, Li Erran Li, Tiancheng Lou, and Jishen Zhao. The dissertation author was the primary investigator and author of these materials.

Chapter 4 contains material from "Suraksha: A Framework to Analyze the Safety Implications of Perception Design Choices in AVs", by Hengyu Zhao, Siva Kumar Sastry Hari, Timothy Tsai, Michael B. Sullivan, Stephen W. Keckler, Jishen Zhao, which appears in International Symposium on Software Reliability Engineering (ISSRE), Oct. 2021. The dissertation author was the primary investigator and author of this material.

VITA

2013 Bachelor of Science, Xi'an Jiaotong University

2016 Master of Science, Xi'an Jiaotong University

2021 Doctor of Philosophy, University of California San Diego

PUBLICATIONS

Hengyu Zhao, Siva Kumar Sastry Hari, Timothy Tsai, Michael B. Sullivan, Stephen W. Keckler, Jishen Zhao, "Suraksha: A Framework to Analyze the Safety Implications of Perception Design Choices in AVs", *International Symposium on Software Reliability Engineering (ISSRE), Oct. 2021*.

Hengyu Zhao, Pingfan Meng, Yubo Zhang, Michael Wu, Haolan Liu, Tiancheng Lou, Jishen Zhao, "Leveraging Routing Information to Enable Efficient Memory System Management of ScanMatch in Autonomous Vehicles", under submission.

Hengyu Zhao, Siva Kumar Sastry Hari, Timothy Tsai, Michael B. Sullivan, Stephen W. Keckler, Jishen Zhao, "Suraksha: A Quantitative AV Safety Evaluation Framework to Analyze Safety Implications of Perception Design Choices", *International Workshop on Safety and Security of Intelligent Vehicles, June 2021 (Co-located with DSN 2021).*

Hengyu Zhao, Yubo Zhang, Pingfan Meng, Hui Shi, Li Erran Li, Tiancheng Lou, Jishen Zhao. "Driving Scenario Perception-Aware Computing System Design in Autonomous Vehicles", *IEEE International Conference on Computer Design (ICCD)*, Oct. 2020. (Best paper in track)

Hengyu Zhao, Yubo Zhang, Pingfan Meng, Hui Shi, Li Erran Li, Tiancheng Lou, Jishen Zhao. "Safety Score: A Quantitative Approach to Guiding Safety-Aware Autonomous Vehicle Computing System Design", *IEEE Intelligent Vehicles Symposium*, Oct. 2020.

Sabur Baidya, Yu-Jen Ku, Hengyu Zhao, Jishen Zhao, Sujit Dey. "Vehicular and Edge Computing for Emerging Connected and Autonomous Vehicle Applications", *ACM/IEEE Design Automation Conference (DAC)* , July 2020. (Invited Paper)

Saransh Gupta, Mohsen Imani, Hengyu Zhao, Fan Wu, Jishen Zhao, Tajana Rosing. "Implementing Binary Neural Networks in Memory with Approximate Accumulation", In *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, Aug. 2020.

Hengyu Zhao, Yubo Zhang, Pingfan Meng, Hui Shi, Li Erran Li, Tiancheng Lou, and Jishen Zhao. "Towards Safety-Aware Computing System Design in Autonomous Vehicles", In *arXiv repo*

Hengyu Zhao, Jiawen Liu, Matheus A. Ogleari, Dong Li, Jishen Zhao. "Processing-in-Memory for Energy-efficient Neural Network Training: A Heterogeneous Approach", *International Symposium on Microarchitecture* (Micro), Oct. 2018.

Hengyu Zhao, Jishen Zhao. "Leveraging MLC STT-RAM for Energy-efficient CNN Training", *Annual International Symposium on Memory Systems* (MEMSYS), Oct. 2018.

Hengyu Zhao, Linuo Xue, Ping Chi, Jishen Zhao. "Approximate Image Storage with Multi-level Cell STT-MRAM Main Memory", *International Conference On Computer Aided Design* (ICCAD), Nov. 2017.

Hengyu Zhao, Colin Weinshenker, Mohamed Ibrahim, Adwait Jog, Jishen Zhao. "Layer-wise Performance Bottleneck Analysis of Deep Neural Networks", *Workshop on Architectures for Intelligent Machines* (AIM), Sep. 2017.

ABSTRACT OF THE DISSERTATION

Architecture and System Support for Safety-aware Autonomous Vehicle Design

by

Hengyu Zhao

Doctor of Philosophy in Computer Science (Computer Engineering)

University of California San Diego, 2021

Professor Jishen Zhao, Chair

Recently, autonomous vehicle development ignited competition among car makers and technical corporations. Low-level automation cars are already commercially available. However, the high automated vehicle where the vehicle drives by itself without human monitoring is still at infancy. Such autonomous vehicles (AVs) rely on the AV system to ensure safety. The AV system consists of two key components: data centers and onboard systems. Data centers are responsible for training deep neural network models, which will be used in the onboard system. It is necessary for data centers to train models efficiently within limited periods. The AV onboard system acts as human drivers, as it monitors surroundings and plans a route for the AV to drive. To ensure safety, the AV onboard system needs to make timely and appropriate driving decisions.

Moreover, a safety validation stage for the onboard system is also required to guarantee AVs' safe operations.

To address the above mentioned challenges, this dissertation proposes three designs. In Chapter 2, this dissertation proposes a processing-in-memory architecture to accelerate deep neural network training stage, which reduces the data movement overhead and improves energy efficiency. In Chapter 3, this dissertation presents the safety score, a latency based safety metric, and the latency model, that represents the correlation between perception latency and surrounding obstacle distribution; then it presents the resource management scheme to optimize safety and performance. In Chapter 4, this dissertation presents Suraksha, a generalized safety validation framework that includes a set of safety metrics and driving scenarios; it also employs Suraksha to perform a case study, where the perception module is studied and safety effects can be collected and analyzed by adjusting perception design choices.

Chapter 1

Introduction

We are on the cusp of a transportation revolution where the autonomous vehicles (also called self-driving cars, uncrewed cars, or driverless cars) are likely to become an essential mobility option [92]. Autonomous vehicles (AVs) that perform some or all driving tasks by themselves are appearing on our roads due to the continuing evolution of automotive and computer technologies [76, 80, 88]. The time when people can sit back and tell their cars where to drive themselves is getting closer.

The state-of-the-art AV system consists of two components: data centers and onboard systems [93]. The AV data center, like conventional data centers, employs massive compute devices to train deep neural network (DNN) models and provide storage spaces for storing data like high definition (HD) maps. In the onboard system, a high automation AV (e.g., level 4) employs a set of sensors to sense the surroundings, various software algorithms to process the sensor data and make driving decisions and a hardware platform that executes the algorithms. The AV onboard system performs similar tasks as a human driver of a conventional car, including perception, localization, and planning control tasks [93].

1.1 Motivation

1.1.1 Safety

AV products must meet stringent safety requirements [69, 102, 118, 150]. AV safety issues stem from numerous sources, such as system design defects, software bugs, or reckless road users [37]. To avoid safety hazards, industrial AV technology developers recently proposed several design standards [70, 71] and safety models [106, 129]. As part of the AV design process, the hazard analysis and risk assessment (HARA) is performed to drive the end-to-end safety requirements, which are then converted to component-level requirements [69, 101]. This dissertation highlights the AV safety challenges in AV data centers and onboard systems, and presents designs to address these problems.

1.1.2 Data Center

Data centers are responsible for storing HD maps and training DNN models for AVs. DNN models employ increasingly large size of parameters and data sets. For example, VGG [131] and AlexNet [82] employ 138M and 61M parameters for image classification, respectively. Training such complex models demands immense computation and memory resources, energy and time. A typical model training period lasts for several days or even weeks [65]. Moreover, reported by Intel, an AV generates more than 4000GB driving data per day [2], which introduces heavy computation workloads for data centers, and presents challenges if the AV fleets keep scaling up.

Moreover, for data centers, it is essential to efficiently train DNN models within a limited period. For example, it is necessary to finish training tasks before the AV fleets upload the new collected data everyday. Otherwise, safety issues may be introduced: AV systems are not updated timely, so they may not make a safe decision in the same scenario met in the past. In this step, the training stage is mainly bounded by low energy efficiency and long training time. Therefore, to process such huge volume of training data within a day, a more efficient

approach to perform DNN training is urgent to be proposed. One critical energy and performance bottleneck when training DNNs is data movement in data centers [147]. As DNN models are becoming deeper and larger, the data volume and the pressure on the runtime system to support data intensive operations substantially increase. Existing research efforts use low-precision data [52] or prune DNN models [143] to address this challenge. Yet, these efforts impose the difficulty of quantifying the impact of model simplification on DNN model accuracy, and they do not fundamentally address the data movement problem in DNN model training.

1.1.3 Onboard System

AV onboard systems are responsible for handling all scenarios where the AV may encounter, and make appropriate driving decisions to drive the AV safely. Therefore, when AVs are operating, AV safety fully relies on the onboard system. As a consequence, the U.S. Department of Transportation recently released "A Vision for Safety" [105] to offer a path forward for the safe deployment of AVs and encourage ideas that deliver safer autonomous driving. Safety remains one of the most critical requirements for autonomous driving system design. To meet critical safety requirement, it is necessary to complete all driving task computations within a limited period [92]. Thus, the onboard system latency makes a difference in AV safety.

Effective safety validation and evaluation is required for the onboard system. While component-level testing is performed, no clear process exists to validate these requirements and analyze the effect they have on AV safety. Without a methodology or framework to analyze the sensitivities of these component-level requirements on AV safety, the designed system may be over-provisioned for some components but compromise the overall functionality (by not providing sufficient resources for some tasks).

1.2 Dissertation Statement

Developing safe AV systems are challenging, and the challenges exist in AV data centers and onboard systems. This dissertation summarizes three key challenges that degrade AV safety:

- The neural network model training is energy and time consuming, so it is difficult to scale up the training scale from the data center side. In the training stage, the data movement between the main memory and processors is the main reason that impedes training energy efficiency and performance.
- The perception module in the AV onboard system is one of the most compute-intensive task; however, the onboard system must meet the real-time requirement under any circumstances, even emergencies. Therefore, it is necessary to identify the factors that may result in long perception latency, so that the end-to-end performance of the onboard system can be optimized.
- The safety validation of onboard systems is a must before any AV products are commercialized, so that AV developers understand what safety effects may be introduced. However, there is no well defined AV safety validation framework that analyzes the safety effects due to various AV configurations and driving scenarios.

In order to address the above mentioned three challenges, this dissertation proposes the following methodologies:

- This dissertation proposes a software/hardware co-design of heterogeneous processing-in-memory (PIM) system: (1) the hardware design incorporates hundreds of fix-function arithmetic units and ARM-based programmable cores on the logic layer of a 3D die-stacked memory to form a heterogeneous PIM architecture attached to CPU; (2) the software design offers a programming model and a runtime system that program, offload and schedule various DNN training operations across compute resources provided by CPU and the heterogeneous PIM.
- This dissertation proposes the "safety score" as the primary metric for measuring the level of safety in AV onboard system design, and the perception latency model, which helps architects estimate the safety score of given architecture and system design without physically testing them in an AV. This dissertation also demonstrates the use of the safety score and latency model, by developing and evaluating an AV computing resource management scheme.
- This dissertation proposes an automated AV safety evaluation framework called Suraksha to

quantify and analyze the sensitivities of different design parameters on AV system safety on a set of driving situations. Suraksha can be used to analyze the safety effects of modulating a set of perception parameters (perception being the most resource demanding AV tasks) on an industrial AV system.

1.3 Dissertation Organization

This dissertation includes five chapters. Chapter 1 introduces safety problems in current AV systems, including data center inefficiencies and onboard system safety requirements. Chapter 2 presents a processing-in-memory architecture to efficiently train DNN models. Chapter 3 proposes the safety score to represent AV safety level by considering onboard system latency, and presents a safety-aware onboard system design to optimize the perception module. Chapter 4 presents Suraksha, which is a generalized AV safety validation framework, and presents a case study, where Suraksha is employed to quantify AV safety level. Chapter 5 concludes this dissertation. All related references are listed in the publication section.

Chapter 2

Efficient DNN Training with Processing-in-Memory Based Data Centers

2.1 Introduction

Recent development of processing-in-memory (PIM) techniques have been explored as a promising solution to tackle the data movement challenge in data centers [48, 77]. We profile various neural network (NN) training workloads and reveal that such workloads have diverse memory access patterns, computation intensity, and parallelism (Section 2.2). As a result, NN training can significantly benefit from heterogeneous PIM – which incorporates both fixed-function logic and programmable cores in the memory – to achieve optimal energy efficiency and balance between parallelism and programmability. However, such heterogeneous PIM architecture introduces multiple challenges in the programming method and runtime system.

First, programming PIMs to accelerate NN training is non-trivial. Today, the common machine learning frameworks, such as TensorFlow [25], Caffe2 [73], heavily rely on a variety of implementations for NN operations on various hardware, and use a middleware to integrate those operations to provide hardware transparency to the user. Such a software design can place high burden on system programmers, because of the increasing hardware heterogeneity and difficulty for program maintenance. Most previous PIM software interfaces [48, 58, 77] require programmers to have the detailed knowledge of underlying hardware. In order to improve productivity and ease-of-adoption of PIM-based NN training accelerators, we need to develop a

programming method that maximizes code reuse without asking the programmer to repeatedly program on different PIMs.

Second, combining fixed-function logics and programmable cores in PIM further complicates the software design. Fixed-function and programmable PIMs employ vastly different programming models: Fixed-function PIMs employ ISA-level instructions accessed via assembly-level intrinsics or via library calls; Programmable PIMs employ standard programming paradigms, such as threading packages or GPGPU programming interfaces [94]. As discussed in recent studies [94], most previous PIM designs adopt homogeneous PIM architectures – with either fixed-function or programmable PIMs – which allows a simplified software interface design. But with heterogeneous PIM, it is critical to design a unified programming model that can accommodate both PIM components.

Finally, the scale of operations in NN training can lead to unbalanced hardware utilization. Ideally, we want to achieve high utilization of PIMs without violating the dependency requirement among NN training operations, by exploiting abundant operation-level parallelism across the host processor and PIMs. However, it can be difficult to achieve so in such a heterogeneous system by pure hardware scheduling, because of the complexity of tracking operation dependency and synchronization. Furthermore, NN training typically adopts a large amount (e.g., tens of thousands) of iterative steps and hundreds of operations per step. Operation dependency across the massive amount of steps and operations can impose synchronization overhead and decrease hardware utilization, when operations are running on multiple computing devices.

The goal of this chapter is to design a PIM-based NN training acceleration system that can efficiently accelerate unmodified training models written on widely-used machine learning frameworks (e.g., TensorFlow). To achieve the goal, we propose a software/hardware co-design of a heterogeneous PIM framework. Our design consists of three components. First, we adopt a heterogeneous PIM architecture, which integrates both fixed-function logics and programmable cores in 3D die-stacked main memory. Second, we extend the OpenCL [4] programming model to address the programming challenges. The programming model maps the host CPU

and heterogeneous PIMs onto OpenCL's platform model and extends OpenCL's execution and memory models for efficient runtime scheduling. Finally, we propose a runtime system, which maximizes PIM hardware utilization and NN-operation-level parallelism. This chapter makes the following contributions:

- We develop a profiling framework to characterize NN training models written on TensorFlow. We identify the heterogeneity requirements across the operations of various NN training workloads. Based on our profiling results, we identify opportunities and key challenges in the software design for efficiently accelerating NN training using PIMs.
- We develop a heterogeneous PIM architecture and demonstrate the effectiveness of such an architecture for training NN models.
- We propose an extension to OpenCL programming model in order to accommodate the PIM heterogeneity and improve the program maintainability of machine learning frameworks.
- We propose a runtime system to dynamically map and schedule NN operations on heterogeneous PIM, based on dynamic profiling of NN operations.

2.2 Background and Motivation

We motivate our software/hardware coordinated design by discussing the challenges of accelerating machine learning training workloads. We employ three widely used CNN training models – VGG-19 [131], AlexNet [82], and DCGAN [115] – as examples in this section. However, our observations can also be applied to various other training workloads (Section 2.6).

2.2.1 NN Training Characterization

In order to understand the characteristics of NN training workloads, we develop a profiling framework (Figure 2.1) by leveraging TensorBoard [24] and Intel VTune [21] to collect software and hardware counter information of training operations. Measuring the number of main memory accesses of individual operations during training can be inaccurate due to the extra cache misses imposed by simultaneously executing operations. As such, we disable inter-operation parallelism

Table 2.1. Operation profiling results for three neural network models. "CI"= computation intensive; "MI"=memory intensive.

VGG-19					
Top 5 CI Ops	Execution Time(%)	#Invocation	Top 5 MI Ops	#Main Memory Access(%)	#Invocation
1. Conv2DBackpropFilter	40.15	16	1. Conv2DBackpropFilter	42.52	16
2. Conv2DBackpropInput	32.68	15	2. BiasAddGrad	35.68	16
3. BiasAddGrad	11.92	16	3. Conv2DBackpropInput	21.06	15
4. Conv2D	10.34	16	4. MaxPoolGrad	0.22	16
5. MaxPoolGrad	1.49	16	5. Relu	0.14	19
Other 13 ops	3.37	232	Other 13 ops	0.38	229
AlexNet					
Top 5 CI Ops	Execution Time(%)	#Invocation	Top 5 MI Ops	#Main Memory Access(%)	#Invocation
1. Conv2DBackpropFilter	33.64	5	1. BiasAddGrad	44.64	3
2. Conv2DBackpropInput	33.46	4	2. Conv2DBackpropInput	36.61	4
3. MatMul	13.54	6	3. Conv2DBackpropFilter	14.79	5
4. Conv2D	10.48	5	4. Relu	1.20	8
5. BiasAddGrad	4.62	3	5. Conv2D	0.46	5
Other 13 ops	4.26	121	Other 13 ops	2.30	119
DCGAN					
Top 5 CI Ops	Execution Time(%)	#Invocation	Top 5 MI Ops	#Main Memory Access(%)	#Invocation
1. Conv2DBackpropFilter	19.98	4	1. Conv2DBackpropFilter	37.21	4
2. Conv2DBackpropInput	17.18	4	2. Conv2DBackpropInput	28.09	4
3. MatMul	14.28	12	3. Slice	17.18	14
4. Conv2D	10.53	4	4. Conv2D	5.45	4
5. Mul	9.89	84	5. Mul	2.22	84
Other 47 ops	28.14	821	Other 47 ops	9.85	819

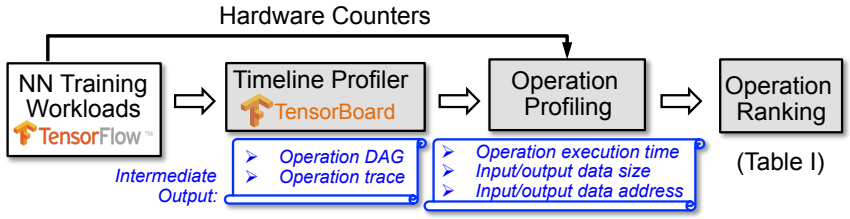


Figure 2.1. Our profiling framework for profiling NN training workloads in TensorFlow.

to ensure characterization accuracy of individual operations.

Table 2.1 illustrates our profiling results of top five most time-consuming and memory-intensive operations, respectively, with three training models. Each model has tens of different types of operations and requires thousands of iterative steps to train; In each step, each type of operation can be invoked up to tens of times. We only show results within one training step, but the characteristics remain stable across training steps.

We make three key observations. First, only several operations dominate training execution time. For example, top five operations in VGG-19 model consume over 95% of total

execution time. Second, the most time-consuming operations are also the most memory intensive. In fact, the top five most time-consuming operations contribute to over 98% of total main memory accesses across all three models. We further classify operations into four classes, shown in Figure 2.2. The first class of operations is compute intensive, and does not have to be offloaded to PIMs, but we can offload them when there are idling hardware units in PIMs. The second class of operations is our target to offload to PIMs. The third class is unusual, and the fourth class does not have big performance impact on model training. The above two observations motivate us to adopt a PIM architecture to accelerate NN training in order to reduce data movement between the host processor and the main memory.

Execution Time	Memory Access %	Example Operations
Long	Low	*Conv2D* in VGG-19
Long	High	*Conv2DBackpropFilter* in VGG-19
Short	High	*Slice* in DCGAN
Short	Low	*Reshape* in AlexNet

Figure 2.2. Four categories of NN training operations.

Third, time-consuming and memory-intensive operations require heterogeneous computation types. It appears that many of such operations are multiplication and addition (e.g., MatMul) or can be decomposed so (e.g., Conv2D). This is inline with previous works on machine learning acceleration [48, 77]. Yet, significant amount of top time-consuming and memory-intensive operations cannot simply be implemented by pure multiplication and addition. For instance, Relu is an activation function that incorporates conditional statement; MaxPool is a sample-based discretization process; ApplyAdam is a first-order gradient-based optimization of stochastic objective functions. Complex operations, such as Conv2DBackpropFilter and Conv2DBackpropInputs, include other logic and computations beyond multiplication and addition. Such non-multiply-add operations can consume over 40% of total execution time. Furthermore, studies on modern multi-tenancy [41] and multimodel training [20] workloads also demonstrate such heterogeneous computation requirement. *This observation motivates us to adopt a heterogeneous PIM architecture that combines fixed-function logic and programmable*

cores.

Most previous works on PIM adopt either fixed-function [48] or programmable [77] computation components in the logic layer of 3D die-stacked memory. In the following, we discuss feasibility, challenges, opportunities of accelerating NN training with software/hardware co-design of heterogeneous PIM.

2.2.2 Feasibility of Heterogeneous PIM Architecture

The logic layer of 3D memory stacks has area, power, and thermal limitations. But previous studies demonstrated the feasibility of adopting both fixed-function and programmable PIMs, while meeting these constraints [155]. We adopt similar methodologies to ensure the feasibility of our architecture implementation (Section 2.4).

2.2.3 Software Design Challenges and Opportunities

There are three challenges for the software design (introduced in Section 2.1): (1) How do we enable high productivity of system programmers and ease-of-adoption of PIM-based NN training accelerators? (2) How do we develop a unified programming model that can efficiently accommodate the host processor, fixed-function PIMs, and programmable PIMs? (3) How do we balance hardware utilization at runtime?

One candidate baseline programming model is OpenCL [4], which is widely used in accelerator-based heterogeneous computing platforms (e.g., GPU and FPGA). We adopt OpenCL, due to its portability, expressiveness, and ability to enable high programming productivity to support programming on heterogeneous systems (details are discussed in Section 2.3.2). However, it is not straightforward to adopt OpenCL for NN model training on the heterogeneous PIM architecture. (1) How do we map the platform model of OpenCL to the heterogeneous PIM architecture? (2) Given the execution model of OpenCL with limited considerations on hardware utilization, how do we make the best use of CPU (the host processor) and different types of PIMs? (3) Given the memory model of OpenCL with limited considerations on synchronization

between hardware units, how do we meet the requirement of frequent synchronizations from NN operations?

Trade-offs between parallelism and programmability. Fixed-function PIMs typically offer high computation parallelism by executing fine-grained, simple operations distributed across massive amount of logic units. However, they are less flexible than programmable PIMs that can be programmed to accommodate a large variety of operations. Furthermore, fixed-function PIMs can impose high performance overhead by (i) frequent operation-spawning and (ii) host-PIM synchronization. Programmable PIMs typically execute coarse-grained code blocks with less frequent host-PIM synchronization. However, the limited number of computational units in programmable PIMs can lead to much lower parallelism than in fixed-function PIMs.

Opportunities in runtime system scheduling. Substantial opportunities exist in leveraging system-level software to optimize resource sharing among various system components. The heterogeneity of our architecture introduces requirements on scheduling model-training operations across the host processor (CPU), fixed-function PIMs and programmable PIMs, based on the dynamic utilization of compute resources on these system components. Yet, we observe that NN training workloads tend to have repeatable (hence predictable) computation behavior over the execution time. As such, system software can accurately predict and dynamically schedule the operations by profiling the resource utilization of various compute elements in the first few steps of modeling training. Such dynamic profiling-based scheduling can achieve the best utilization of computation resources, while improving energy efficiency.

2.2.4 CPU vs. GPU – Where to Attach Heterogeneous PIMs?

Today, NN-training workloads can be executed on both CPU- and GPU-based systems. Recent silicon interposer technology allows both types of systems to adopt 3D die-stacked memories closely integrated with logic components. For example, modern GPU device memories [23] are implemented by high-bandwidth memory technology. High-end CPU servers integrate high-bandwidth memories using the DRAM technology adopted from hybrid memory

cubes.

Our heterogeneous PIMs are logic components closely integrated with die-stacked memories. Therefore, they are generally applicable to both CPU or GPU systems. However, this work focuses on the software design for heterogeneous PIMs attached on CPU systems, due to the constraint of current GPU systems. Today, GPU systems often fuse and organize computation kernels into NN layers rather than fine-grained operations, because of the inefficiency of compute preemption and thread scheduling. This significantly limits the flexibility of operation scheduling on GPU.

The NVIDIA Volta GPU provides certain support for fine-grained acceleration of NN training operations, yet only available with limited number of threads. Modern CPU systems are easy to access and program; this enables easy-to-adopt and flexible programming abstraction and system library functions.

2.3 Design

To address the aforementioned challenges, we propose a software/hardware co-design of heterogeneous PIM framework to accelerate NN training. Our design consists of a heterogeneous PIM architecture, an extended OpenCL programming model, and a runtime system. Figure 2.3 depicts our architecture configuration. Figure 2.4 shows the process of building and executing NN training with our software framework. Given an OpenCL kernel to implement an operation, our system extracts code sections from the kernel and compiles them into a set of binaries to run on CPU, programmable PIM, and fixed-function PIMs, respectively. After the training workload starts to execute, our runtime scheduler profiles the first step of training to obtain operation characterization. It then performs dynamic scheduling of operations across CPU, programmable PIM, and fixed-function PIMs in the rest of training steps. Our runtime system incorporates two key components: (i) an operation-pipeline scheme, which allows multiple NN operations to co-run on PIMs to improve hardware utilization and (ii) a recursive operation-execution scheme,

which allows the programmable PIM to call fixed-function PIMs to improve hardware utilization and avoid frequent synchronization between CPU and PIMs.

Software/hardware co-design principles. Our software design supports our hardware configuration in the following manner. First, our software design offers a portable programming model across the host processor, fixed-function PIMs, and the programmable PIM. Our programming model provides a unified abstract to program various PIMs, which need to be programmed in separate manners in conventional systems. Our runtime scheduling scheme effectively optimizes PIM hardware utilization. Our runtime system also enables recursive calls between the programmable PIM and fixed-function PIMs. Our architecture design supports our software design in two ways: our heterogeneous PIM architecture enables efficient NN training acceleration by exploiting the heterogeneous characteristics of software operations; We employ a set of hardware registers to track PIM hardware utilization information, which is required by our runtime scheduling.

2.3.1 Heterogeneous PIM Architecture

To accommodate various types of operations that are likely to execute on PIMs, we adopt a heterogeneous PIM architecture consisting of (i) a programmable PIM, which is an ARM core and (ii) massive fixed-function PIMs, which are adders and multipliers distributed across all memory banks. While our design can be used with various 3D die-stacked memory devices, we employ a 32-bank memory stack (where a bank is a vertical slice in the stack) as an example. Figure 2.3 depicts our architecture configuration. Section 2.4 describes hardware implementation details.

2.3.2 Programming Model for Heterogeneous PIM

We extend the OpenCL programming model to program the heterogeneous PIM. OpenCL has been widely employed to enable program portability across accelerator-based, heterogeneous computing platforms (e.g., GPU and FPGA). We use OpenCL because of the following reasons.

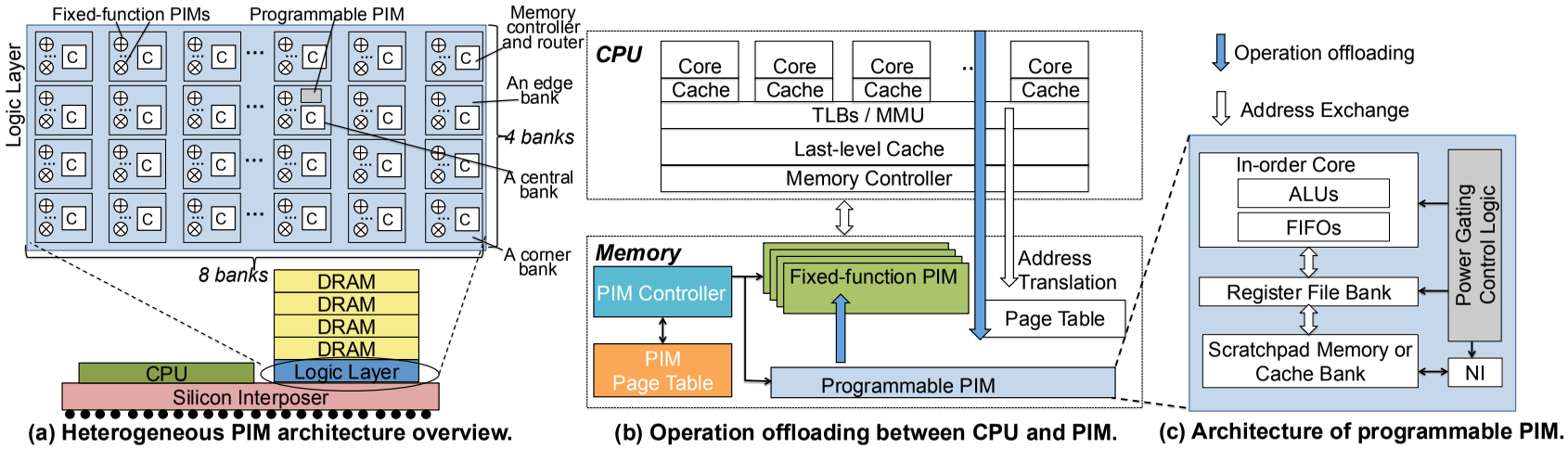


Figure 2.3. Architecture overview of the proposed heterogeneous PIM.

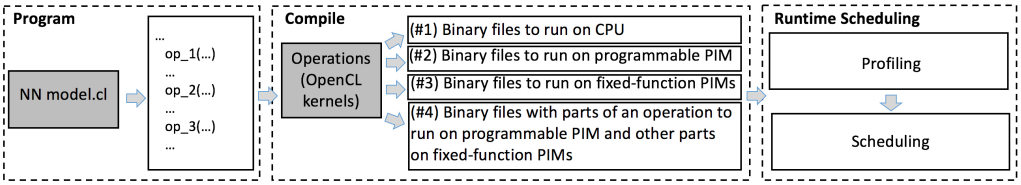


Figure 2.4. The process of executing NN training with our software framework design.

First, by treating the fixed-function PIMs and programmable PIM as accelerators, the semantics of OpenCL naturally fit into the heterogeneous PIM environment. Second, writing a program for the heterogeneous PIM based on an abstract and unified hardware model in OpenCL, the programmer can write the program just once but run it on a variety of PIMs. Therefore, by using OpenCL, we can hide hardware variety of the heterogeneous PIM from system programmers, improve their productivity, and enable code portability.

Other programming models, such as OpenACC [3, 7] and OpenMP [57], can also hide hardware heterogeneity and reduce programmers' burden. However, these are higher-level programming models, which rely on compilers to transform programs into a lower-level programming model, such as OpenCL, to enable code portability. We focus on OpenCL in our study, because it provides a foundation for those higher-level programming models.

Overview of our programming model. Table 2.2 summarizes our extension to OpenCL. Our platform model includes multiple types of heterogeneous devices. Such platform model is driven by the characteristics of NN training operations. Our execution model adds (i) recursive kernel invocation to enable kernel invocation between PIMs to support complex NN operations (e.g., Conv2DBackpropFilter) and (ii) operation pipeline to improve hardware utilization for small NN operations with limited parallelism (e.g., Slice). Finally, we extend the memory model to support a single global memory shared between the host processor and accelerators. We also add explicit synchronization across different PIMs and CPU (host processor) to enforce execution orders across NN operations.

OpenCL background. The existing OpenCL adopts a host-accelerator platform model as shown in Figure 2.5(a). A host processor connects to one or more compute devices (i.e.,

Table 2.2. OpenCL for the heterogeneous PIM.

	Native OpenCL	**Extensions for Heterogeneous PIM**
Platform model	Host + accelerators (e.g., host + GPU).	Host + two types of accelerators (fixed-function PIMs and programmable PIM) driven by the characteristics of NN training.
Execution model	Host submits work to accelerators.	• Host submits work to accelerators; • Accelerators submit work to accelerators (i.e., recursive kernel invocation); • Work execution pipeline (i.e., operation pipeline); • Work scheduling based on dynamic profiling.
Memory model	• Multiple types of memory with a relaxed consistency model; • The global memory is not shared; • No defined synchronization across accelerators.	• A single global memory with a relaxed consistency model; • The global memory is shared; • Explicit synchronization across PIMs and CPU.

accelerators). A compute device is divided into one or more compute units, each of which is further divided into one or more processing elements (PE). An OpenCL program consists of kernels for compute devices and a host program. The host program runs on CPU and enqueues commands to a command-queue attached to a compute device.

In order to employ OpenCL programming model on the heterogeneous PIM system, we investigate how to map the heterogeneous PIM system onto the OpenCL model, and extend the OpenCL model for efficient runtime scheduling. In the following, we discuss our mapping method from the perspectives of platform model, execution model, and memory model. Table 2.2 summarizes our programming model extension.

Heterogeneous PIM platform model. Figure 2.5(b) illustrates our platform model. A large number of fixed-function PIMs provide massive parallelism for data processing. Each fixed-function PIM is a PE (in the OpenCL jargon). All fixed-function PIMs in all memory banks form a compute device. All fixed-function PIMs in a bank form a compute unit. Each programmable PIM is a compute device; each core of the programmable PIM is a PE. Hence,

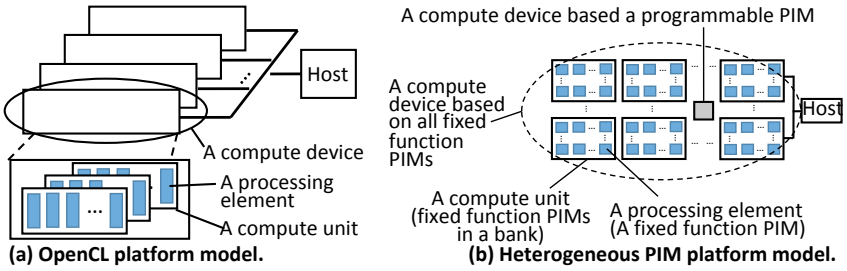


Figure 2.5. Enabling OpenCL platform model on heterogeneous PIM systems.

within the context of OpenCL, a heterogeneous PIM system has heterogeneous compute devices. We expose fixed-function PIM and programmable PIM as distinct compute devices to give control flexibility to the runtime system for operation scheduling. An OpenCL operation can be offloaded to any compute device that supports the operation execution.

Execution model. Tasks (i.e., operations in NN model training) to be launched on any PIM are represented as kernels managed by a host program, as in a traditional OpenCL program. If the task includes instructions that cannot be executed on the fixed-function PIM, then the task will not be scheduled by the OpenCL runtime to run on the fixed-function PIM. Otherwise, a task can run anywhere (CPU, fixed-function PIM, and programmable PIM). The OpenCL runtime (on CPU) is in charge of task scheduling between different PIMs and CPU. Leveraging low-level APIs (Section 2.4.1) and hardware registers, the runtime can determine whether a specific PIM is busy and whether a specific task is completed. We describe the scheduling algorithm in Section 2.3.3. Binary files for a task to run on CPU, fixed-function PIM, or programmable PIM are generated during the compilation stage. Given an OpenCL kernel for a task, we generate four binary files as shown in Figure 2.4. Section 2.4 discusses details of binary generation.

Binaries (#3) and (#4) in Figure 2.4 allow *recursive PIM kernel*, a new execution scheme for our heterogeneous PIM design. A kernel in the programmable PIM can trigger data processing with fixed-function PIMs. This is supported by the programmable PIM runtime and implemented by calling small kernels loadable on fixed-function PIMs. By combining multiple kernels into a single kernel, the recursive PIM kernel scheme reduces overhead of kernel spawning and

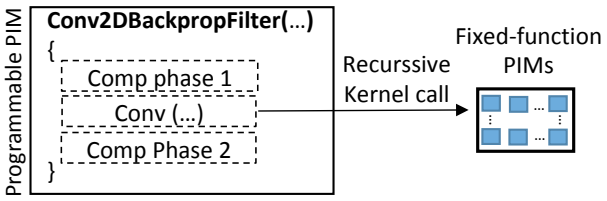


Figure 2.6. An example of the recursive PIM kernel.

synchronization between the host and PIMs. Figure 2.6 shows an example that further explains the recursive PIM kernel. In the example, we illustrate an NN operation, Conv2DBackpropFilter, which is offloaded to the programmable PIM as a kernel; the kernel includes computation phases 1 and 2 that cannot be offloaded to the fixed-function PIMs. Conv2DBackpropFilter includes convolution computation ("Conv(...)" in the figure); The programmable PIM offloads this portion of computation to fixed-function PIM as a smaller kernel. The computation phases 1, 2 and convolution are combined as a single recursive PIM kernel, which reduces the synchronization between CPU and PIMs.

In general, the four binary files provide convenience for scheduling on CPU, the fixed-function PIMs and programmable PIM, and hence allows the runtime to maximize utilization of CPU and PIMs.

Memory model. The existing OpenCL defines four distinct memory regions in a compute device: global, constant, local, and private. On a heterogeneous PIM system, only a single global memory (i.e., the main memory) exists. In addition, the global memory is shared between CPU and PIMs, and addressed within a unified physical address space. This memory model requires synchronization at multiple points: (1) between CPU and PIMs; and (2) between different PIMs. The synchronization is necessary to avoid data race and schedule operations.

To implement effective synchronization, we employ the programmable PIM to drive the synchronization and avoid frequent interrupts to CPU. In particular, for synchronization between CPU and PIMs, the programmable PIM checks the completion of operations offloaded to PIMs (either programmable or fix function PIMs) and sends the completion information to CPU. For

synchronization between different PIMs, the programmable and fix function PIMs synchronize through global variables in main memory.

Between CPU and PIMs, we introduce explicit synchronization points to synchronize the accesses to shared variables. To the host processor, the whole set of fixed-function PIMs or the programmable PIM appear as another processor. We employ standard synchronization schemes (e.g., barriers and locks), similar to the ones in a shared-memory multiprocessor. For fixed-function PIMs, their operations are atomic and the synchronization points are not expected in the middle of operations. For programmable PIMs, the synchronization points can be in the middle of a kernel. This is feasible based on global lock variables shared between CPU and PIMs. To support memory consistency, we adopt a relaxed memory consistency model, which aims to improve performance and reduce hardware complexity. In particular, an update to a memory location by a fixed-function PIM is not visible to all the other fixed-function PIMs at the same time. Instead, the local view of memory from each fixed-function PIM is only guaranteed to be consistent right after the kernel call to fixed-function PIMs. Between the fixed-function PIMs and programmable PIM, we employ the same consistency scheme: updates to memory locations by the entire set of fixed-function PIMs are not visible until the end of the kernel call to the fixed-function PIMs.

Because of our shared memory model, there is no data copy overhead before and after PIM kernel calls. Based on the above synchronization schemes, PIM kernel calls can be launched asynchronously to overlap computation on CPU and PIMs.

Support for easy program maintenance. To use the extended OpenCL programming model, operations need to be re-written using OpenCL. To write OpenCL code for operations, one can use OpenACC directives and compilers [3, 7] to automatically transform the original code into OpenCL code. This can significantly simplify the programming work. Furthermore, the number of operations for machine learning models is limited (tens of operations). Hence, using OpenCL to implement those machine learning operations is feasible. Other than that, however,

the higher level software components (e.g., most of the middleware components, operation APIs, and Python syntax for using machine learning models) remain the same. This enables easy maintenance of machine learning frameworks.

2.3.3 Runtime System Design

Our runtime system is in charge of scheduling operations to fixed-function PIMs, programmable PIM, and CPU. To minimize NN training time, the runtime strives to maximize utilization of PIMs and CPU to optimize system throughput. The runtime schedules operations based on the following two steps.

Step 1: profiling. The runtime profiles performance of all operations on CPU. The profiling happens in only one step of NN model training. NN model training typically has a large amount of iterative steps (thousands and even millions of steps). Using one step for profiling has ignorable impact on performance. In addition, all steps almost have the same classes of operations; performance of operations (particularly execution time and the number of main memory access) remains stable across steps. Therefore, one step is sufficient for profiling purpose. During profiling, the runtime executes operations one by one in CPU, collecting execution time and the number of main memory access level cache misses of each operation with hardware counters. Based on the profiling results in the step, the runtime employs the following algorithm to determine the candidate operations to be offloaded to PIMs.

To determine the candidate operations, the runtime sorts operations into two lists (in descending order) based on execution time and the number of main memory accesses, respectively. Each operation in each of the two lists is correlated to an index, i.e., each operation has two indexes. With each operation, the runtime calculates a global index by adding these two indexes. Based on the global indexes, the runtime sorts operations into a global list. The runtime chooses top operations in the global list to offload to PIMs. Those top operations account for x% of total execution time of one step (x = 90 in our evaluation). The above algorithm is inspired by feature selection process in machine learning [61]. The goal of this algorithm is to select those

Table 2.3. Low-level APIs for PIMs.

Name	Description
int pim_fix(int* pim_ids, void* args, void* ret, size_t num_pim)	Asks specific fixed-function PIMs to work with input arguments `args` and return results `ret` and a work ID.
int pim_prog(int pim_id, pim_program kernel, void* args, int* args_offset, void* ret, size_t ret_size)	Asks a programmable PIM to work on a kernel (an operation) and return a work ID.
int pim_status(int pim_id)	Checks whether a specific PIM is busy.
int work_query(int work_id)	Checks whether a specific operation is completed.
void work_info(int work_id, int* pim_ids, int* data_loc)	Queries the computation location (pim_ids) and input/output data location (i.e, which DRAM banks) for a specific operation.

operations that are both time-consuming and have a large number of main memory accesses.

Step 2: scheduling. Given the candidate operations to offload, the runtime makes the scheduling decision based on the following three principles.

- Scheduling operations to execute on fixed-function PIMs as much as possible.
- Scheduling operations to execute on PIMs (not CPU) as much as possible. In case all fixed-function or programmable PIMs are busy, the runtime will schedule the candidate operations to execute on CPU;
- Scheduling needs to respect data dependency across operations.

The first principle favors fixed-function PIMs over other compute units, because fixed-function PIMs are more energy efficient and typically performs faster with higher parallelism than other compute units. The second principle avoids CPU idling and introduces parallelism between CPU and PIMs. The third principle ensures execution correctness. Each operation defined in the machine learning frameworks typically has explicit input and output data objects (e.g., Tensors in TensorFlow), which provides convenience in tracking data dependencies across operations.

Operation pipeline. The above scheduling algorithm and principles enable operation pipeline to maximize hardware utilization. In particular, when an operation in a step cannot fully utilize fixed-function PIMs, our runtime schedules an operation in the next step to execute a portion of its computation by utilizing idling fixed-function PIMs as long as the two operations

do not depend on each other.

In essence, these two operations can enable a pipelined execution manner. For instance, in AlexNet, a single convolution operation with a filter size of 11×11 consumes 121 multiplication and 120 addition (241 fixed-function PIMs in total). In case we have 444 fixed-function PIMs in total (Section 2.4.4), the utilization of fixed-function PIMs is only 54%. To improve hardware utilization, the runtime can schedule multiplication and addition from an operation (or operations) in the next step to execute on fixed-function PIMs. Once the convolution operation in the current step is completed, the partially executed operation(s) from the next step can immediately utilize the newly released fixed-function PIMs to improve hardware utilization and performance. This indicates that an operation can dynamically change its usage of PIMs, depending on the availability of PIMs. Such dynamic nature of operation execution is feasible based on a runtime system running on the programmable PIM (Section 2.4.3 presents implementation details).

2.4 Implementation

2.4.1 Low-level APIs for PIM Runtime System

We introduce several low-level API functions for fixed-function and programmable PIMs. These API functions allow direct control of individual PIMs, and provide foundation for our runtime. The API achieves the following functionality: (1) offloading a specific operation into specific PIM(s); (2) tracking the status of PIMs, including examining whether a PIM is busy or not; (3) querying the completion of a specific operation; (4) querying the computation location (i.e., which PIM) and input/output data location (i.e, which DRAM banks) for a specific operation. Table 2.3 summarizes our API functions.

2.4.2 OpenCL Binary Generation

To schedule operations to execute on CPU, fixed-function PIMs, or programmable PIM, we generate four binary files (Figure 2.4). In order to generate the binary file (#3) that corresponds

to a portion of a large operation (an OpenCL kernel) to execute on fixed-function PIMs (e.g., the convolution within the operation Conv2DBackpropFilter), we first extract code sections from the corresponding OpenCL kernel. We then transform these code sections into a set of small kernels to execute on fixed-function PIMs. Finally we compile them into binary file (#3). In the original OpenCL kernel, these extracted code regions are replaced with the kernel calls and then compiled into binary file (#4) to execute on the programmable PIM. Binary files (#1) and (#2) are generated during the regular compilation stage.

2.4.3 Runtime Implementation

Our runtime consists of two components, which execute on the CPU and the programmable PIM, respectively.

The runtime on CPU. To support our runtime scheduling, we extend the runtime system of TensorFlow by adding approximately 2000 lines of code. The runtime on CPU schedules operations on CPU and PIMs, based on hardware utilization information provided by the low-level APIs. It does not support the implementation of recursive PIM kernels. In other words, the runtime on CPU is only responsible for offloading a kernel – which can have a part of its computation offloadable to fixed-function PIMs – to the programmable PIM. Our modifications to TensorFlow runtime include (1) device initialization and characterization using OpenCL intrinsics; (2) creating a device context and instance for a PIM device; (3) providing a new OpenCL device abstraction to other components of Tensorflow; (4) a mechanism to communicate with the runtime on the programmable PIM. This is one-time modification to Tensorflow, but can support various PIM hardware configurations without involving system programmers' future efforts.

The runtime on programmable PIM. The runtime on the programmable PIM supports recursive PIM kernels and operation pipeline. In particular, a kernel with a part of its computation replaced with kernel calls to fixed-function PIMs is handled by the runtime on the programmable PIM, which automatically offloads the computation to fixed-function PIMs. In order to keep track

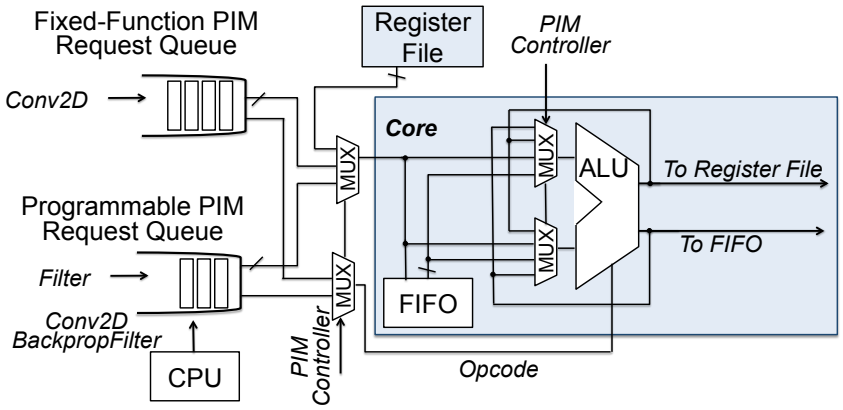


Figure 2.7. Heterogeneous PIM implementation.

of the dynamic utilization of fixed-function PIMs, our runtime on the programmable PIM records the numbers of additions and multiplications already completed in each operation offloaded to the programmable PIM, as well as the remaining additions and multiplications.

2.4.4 Hardware Implementation

Figure 2.3 and Figure 2.7 illustrate our hardware implementation. The programmable PIM employs an ARM Cortex-A9 processor with four 2GHz cores. Each core has an in-order pipeline. In individual NN training models, operations that are potentially offloaded to the programmable PIM (e.g., ApplyAdam, MaxPooling, and ReLU) are typically not executed at the same time. Therefore, we only adopt one programmable PIM in our design. Even if we simultaneously train multiple NN models, the chance of having multiple operations to use the programmable PIM at the same time is low according to our evaluation with mixed workload analysis (Section 2.6.6).

Because a significant portion of NN training operations can be decomposed to addition and multiplications (Section 2.2.1), we implement our fixed-function PIMs as 32-bit floating point multipliers and adders. We implement equal numbers of multipliers and adders in the pool of fixed-function PIMs. Our low-level APIs allow us to map operations to fixed-function PIMs that are in the same bank as input data of the operations. In addition, we accommodate random memory access pattern in NN computation by adopting buffering mechanisms [48].

We determine the fixed-function PIM configurations by employing a set of architectural level area, power, and thermal modeling tools, including McPAT [90] and HotSpot [132], to perform design space exploration of the logic die of 3D DRAM. Based on our study, the total number of allowed fixed-function PIMs is limited by the area of the logic die. With our baseline 3D DRAM configuration (Section 2.5), we can distribute 444 fixed-function PIMs (pairs of multipliers and adders) across the 32 banks in the logic die. It is impossible to distribute these fixed-function PIMs evenly to each bank. We consider the placement of the fixed-function PIMs on 32 banks based on the following policy: we place more fixed-function PIMs on edge and corner banks than on central banks (Figure 2.3 (a)). The rationale behind is that the banks at the edge and corner have better thermal dissipation paths than central banks. Therefore, these banks can support higher computation density.

Furthermore, we employ a set of registers as shown in Figure 2.7. Each register indicates the idling of either a bank of fixed-function PIMs or the programmable PIM. The registers allow our software runtime scheduler to query the completion of any computation and decide the idleness of processing units.

2.5 Experimental Setup

2.5.1 Simulation Framework

In order to evaluate the performance of our design, we model fixed-function PIM and programmable PIM architectures, respectively, using Synopsys Design Compiler [135] and PrimeTime [136] with Verilog HDL. We adopt HMC 2.0 [68] timing parameters and configurations for our evaluation of 3D memory stack. Baseline memory frequency is set to 312.5 MHz, which is the same as HMC 2.0 specification [68]. This is also used as the working frequency of our heterogeneous PIM. We employ a trace generator developed on Pin [95] to collect instruction trace, when running our OpenCL kernel binaries on CPU. We develop a python-based, trace-driven simulation framework based on our design to evaluate the execution time of various

training workload traces. Our simulator also incorporates our runtime scheduling mechanisms.

2.5.2 Power and Area Modeling

We adopt 10nm technology node for the host CPU and the logic die of the PIMs; 25nm technology node for the DRAM dies. We measure CPU and GPU power with VTune [117] and *nvidia-smi*, respectively. Our power model considers whole system power when we evaluate the power of heterogeneous-PIM-based systems, including CPU and the memory stack. We calculate the power and area of the programmable PIM using McPAT [90]. We evaluate the power and area of fixed-function PIMs using Synopsys Design Compiler [135] and PrimeTime [136].

2.5.3 Workloads

We evaluate various training models, including VGG-19 [131], AlexNet [82], Deep Convolutional Generative Adversarial Networks (DCGAN)) [115], ResNet-50 [62], Inception-v3 [137], Long Short Term Memory (LSTM) with dropout [145] and Word2vec [98]. LSTM and Word2vec are evaluated in Section 2.6.6. The rest models are widely used in recent studies on CNN training and image classification.

Training Datasets. We employ ImageNet as training data set of VGG-19, AlexNet, ResNet-50, and Inception-V3. ImageNet is a large image dataset with millions of images belonging to thousands of categories. DCGAN employs MNIST dataset [85]. LSTM adopts Penn Tree Bank (PTB) [145] dataset. Word2vec employs "questions-words" dataset [19] in TensorFlow.

Training framework and batch Size. We adopt TensorFlow [25] as our training framework. We adopt default batch sizes of each training model in TensorFlow. The batch size of VGG-19, AlexNet and Inception-v3 is 32. The batch size of Word2vec and ResNet-50 is 128. DCGAN has a batch size of 64. LSTM employs a batch size of 20.

Table 2.4. System configurations.

CPU	Intel Xeon E5-2630 V3@2.4GHz
Main memory	16GB DDR4
Operating system	Ubuntu 16.04.2
GPU	NVIDIA GeForce GTX 1080 Ti (Pascal)
GPU cores	28 SMs, 128 CUDA cores per SM, 1.5GHz
L1 cache	24KB per SM
L2 cache	4096KB
Memory interface	8 memory controllers, 352-bit bus width
GPU main memory	11GB GDDR5X

2.5.4 Real Machine Configurations

To compare performance and energy efficiency of heterogeneous PIM with GPU and CPU, we run the training models on (1) NVIDIA GeForce GTX 1080 Ti graphic card [22] and (2) CPU listed in Table 2.4. Our GPU-based training evaluations adopt CUDA 8 [6] and NVIDIA cuDNN 6.0 library [107]. GPU utilizations of each training model in TensorFlow are: Inception-v3 (average: 62%; peak: 100%); ResNet-50 (average: 44%; peak: 58%); AlexNet (average: 30%; peak: 34%); VGG-19 (average: 63%; peak: 84%); DCGAN (average: 28%; peak 42%. We use NVIDIA's profiling tool [18] and Intel's VTune [21] to collect performance and power statistics.

2.6 Evaluation

Our experiments compare among the following five configurations, including our design.

- **CPU** – Executing all training operations on CPU;
- **GPU** – Executing all training operations on GPU;
- **Progr PIM** – Programmable PIMs only, which executes all operations on as many ARM-based programmable cores as needed by workloads (without our runtime scheduling);
- **Fixed PIM** – Fixed-function PIMs only, which executes the operations that can be offloaded on fixed-function PIM and other operations on CPU (without our runtime scheduling);
- **Hetero PIM** – Our heterogeneous PIM design (including our runtime scheduling).

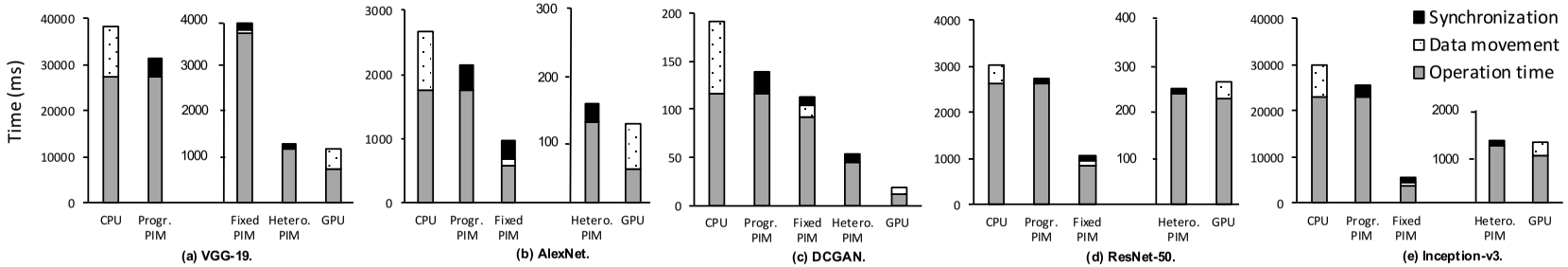


Figure 2.8. Execution time breakdown of five NN models.

2.6.1 Execution Time Analysis

Figure 2.8 shows execution (training) time of various NN training models. We break down the execution time into synchronization time, data movement time and operation time (i.e., computation time in CPU, GPU or PIMs). For GPU-based systems, the data movement time is the time for data transfer between main memory and GPU global memory. Certain amount of data transfer time is overlapped with GPU computation, e.g. by copying a minibatch of images to the GPU memory, while the computation on GPU is processing another minibatch. Our breakdown only shows the data transfer time that is not hidden by the computation. For PIM-based systems, the data movement time is the time for data transfer between CPU and the main memory. Our runtime scheduling allows operations to execute concurrently on CPU and PIMs.

We observe that PIM-based designs (including Fixed PIM, Progr PIM and Hetero PIM) perform much better than CPU, with 19%-28× performance improvement. Compared with Progr PIM and Fixed PIM, our design has 2.5×-23× and 1.4×-5.7× performance improvement, respectively. PIM-based designs also significantly reduce data movement overhead, compared to CPU and GPU. Overall, Hetero PIM leads to the lowest synchronization and data movement overhead among all configurations.

The performance benefit of Hetero PIM stands out with larger training models and larger working sets due to (i) more reduction in data movement and (ii) higher parallelism between host CPU and PIMs introduced by more offloadable operations. DCGAN has smaller model and working set than others. Therefore, Hetero PIM appears to result in worse performance than GPU with DCGAN; yet, compared with other configurations, our design still significantly improves performance. ResNet is a large training model with large working sets. As a result, Hetero PIM leads to better performance than GPU with ResNet. With other training models, Hetero PIM leads to performance close to (within 10% of) GPU. GPU has good performance because of its massive thread-level parallelism. Our design leads to much better performance

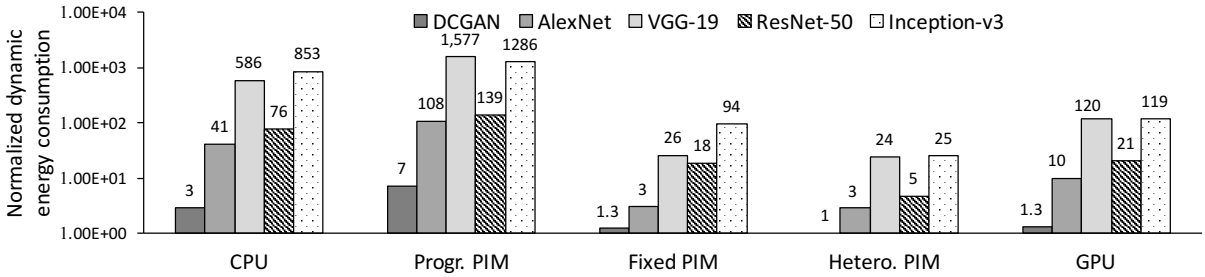


Figure 2.9. Normalized dynamic energy of various NN models.

than all other configurations.

2.6.2 Energy Consumption Analysis

Figure 2.9 shows the dynamic energy consumption of the five NN models with five different configurations. The energy consumption results are normalized to the results of Hetero PIM. We observe substantial energy benefit of using our design: it consumes 3×-24× and 1.3×-5× less energy than CPU and GPU, respectively. CPU consumes higher dynamic energy than Hetero PIM, Fixed PIMs, and GPU, even though its power consumption is the lowest among all of these configurations (note that we take CPU power into account when we calculate the power of PIMs and GPU, in order to evaluate full-system power consumption). This is because CPU has the longest execution (training) time. Furthermore, we notice that the dynamic energy consumption of Progr PIM is higher than that of other configurations, because the speed of Progr PIM is only slightly faster than that of CPU, yet the dynamic power of Progr PIM is higher than that of CPU due to the additional processing units in Progr PIM. Overall, Hetero PIM leads to the lowest dynamic energy consumption across all configurations.

2.6.3 Comparison with Prior PIM-based NN Acceleration

Figure 2.10 shows a quantitative comparison between our design and a recent PIM-based NN accelerator design, Neurocube [77] (qualitative comparison is in Section 2.7). Neurocube also reduces data movement overhead and improves energy efficiency by using PIM technology. However, our work outperforms Neurocube in terms of performance and energy efficiency.

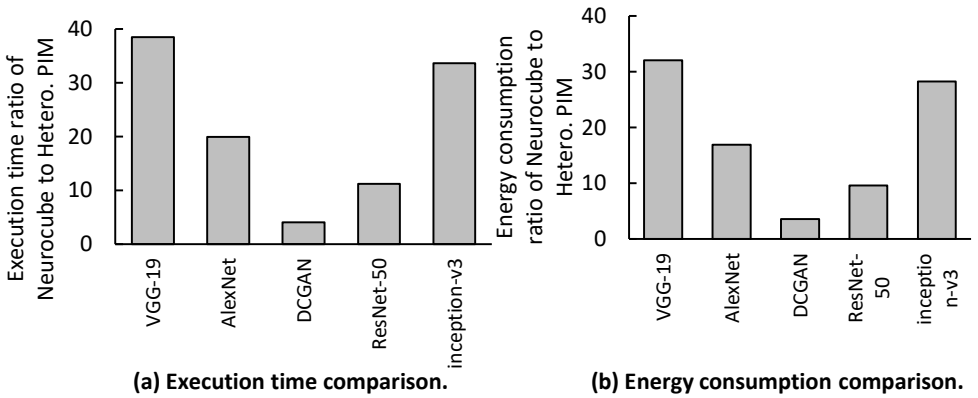


Figure 2.10. Performance and energy comparison with Neurocube.

With highly compute-intensive models, such as VGG-19 and Inception-V3, our design achieves much higher performance and energy-efficiency improvement than Neurocube. Even with less compute-intensive models, such as DCGAN, our work can achieve at least 3× higher performance and energy efficiency than Neurocube. The reason for the improvement is two-fold: (1) Neurocube only adopts programmable PIMs, while our design employs energy-efficient, highly-parallel fixed-function PIMs to accelerate fine-grained operations; (2) Our design employs runtime scheduling that effectively optimizes hardware utilization (evaluated in Section 2.6.5).

2.6.4 Sensitivity Study

Frequency Scaling. We adopt three different frequencies for fixed-function PIMs and programmable PIM: their original frequencies (1×), doubling of their frequencies (2×) and quadrupling of their frequencies (4×). We use a phase-locked loop module to change the frequency. We study execution (training) time with the different frequencies.

Figure 2.11 shows the results. We observe that with higher frequency, the heterogeneous PIM performs better than GPU. With 2× frequency, Hetero PIM performs 36% and 17% better than GPU, with VGG-19 and AlexNet, respectively. With 4× frequency, Hetero PIM performs 37% and 60% better than GPU, with VGG-19 and AlexNet respectively. We also observe that the synchronization and data movement overheads are reduced, when using higher frequencies.

Programmable PIM Scaling. We employ three different configurations for Hetero PIM,

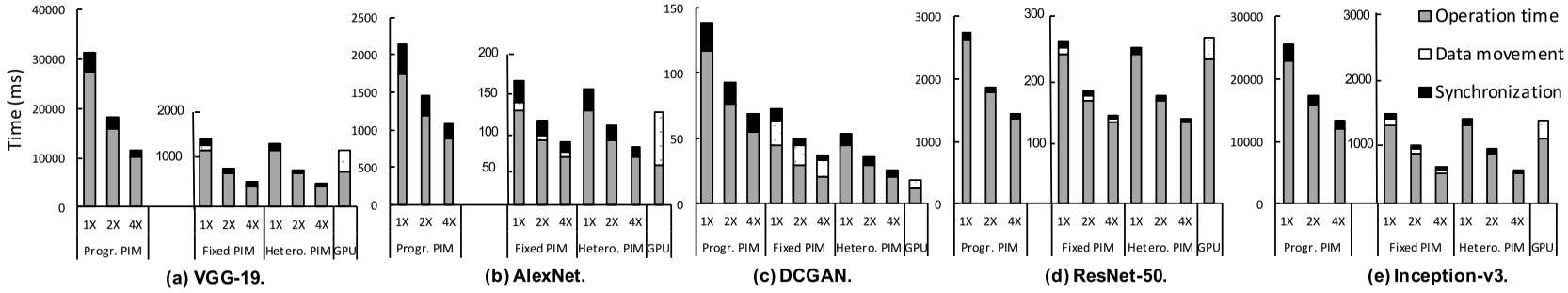


Figure 2.11. Execution time breakdown of various NN models with 3D memory frequency scaling.

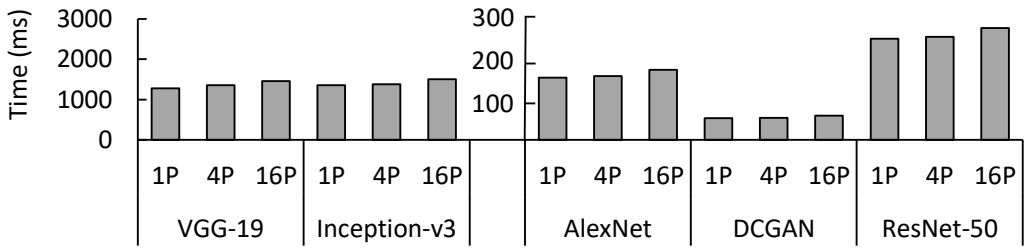


Figure 2.12. Execution time with Progr PIM scaling.

while keeping the area of logic die in the memory stack unchanged. We scale the number of Progr PIM (ARM cores) from one to two to 16, while the rest of the logic die area is used to implement Fixed PIM. The three configurations are labeled as *1P*, *4P* and *16P*, respectively.

Figure 2.12 shows our results. The figure reveals that the performance difference between the three configurations is relatively small. The performance difference between *16P* and *1P* is 12%–14%. The reason is two-fold: (1) One Progr PIM is sufficient for the NN models to schedule and pipeline operations; (2) Using more Progr PIMs loses more Fixed PIMs, given the constant area in the logic layer of memory stacks.

2.6.5 Evaluation of Software Impact

We isolate the the impact of our software (runtime) techniques from that of Hetero PIM hardware. We aim to provide more insightful analysis on the effectiveness of software/hardware co-design. In particular, we study execution time, energy and utilization of Fixed PIM with and without the recursive PIM kernel call (RC) and operation pipeline (OP) – our two major runtime techniques. *Without RC and OP, we also compare Hetero PIM hardware design with Fixed PIM and Progr PIM, in terms of execution time and energy. This comparison allows us to study the impact of Hetero PIM architecture with the absence of our runtime techniques.*

Execution time analysis. As shown in Figure 2.13, Hetero PIM without runtime scheduling performs better than Progr PIM and Fixed PIM by up to 8.5×. This demonstrates the necessity of using Hetero PIM architecture. However, comparing with Fixed PIM, the performance benefit of Hetero PIM hardware is not significant (7%-30%). After incorporating the runtime scheduling

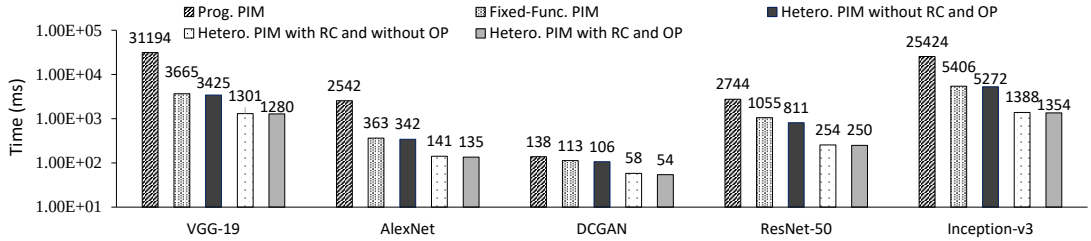


Figure 2.13. Execution time with and without RC and OP.

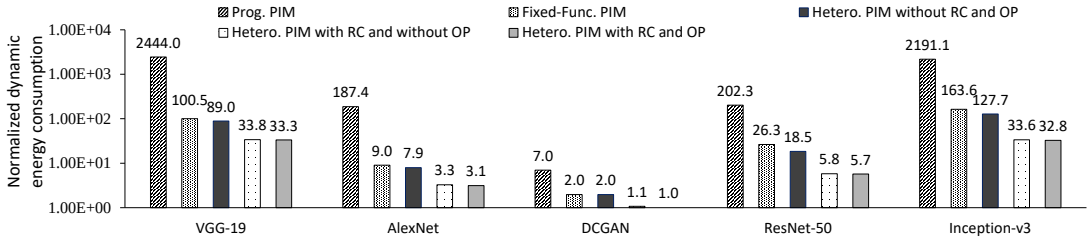


Figure 2.14. Dynamic energy with and without RC and OP.

techniques, the performance of Hetero PIM is improved by up to 3.8×. This result demonstrates the necessity of using an efficient runtime to maximize the benefit of Hetero PIM architecture.

Energy analysis. Figure 2.14 shows our energy results normalized to the energy of Hetero PIM with RC and OP. We have similar observations as the execution time analysis: Hetero PIM without runtime scheduling performs better than Progr PIM and Fixed PIM by up to 2.7×. With RC and OP, we further reduce the energy of Hetero PIM by up to 3.9×.

PIM utilization analysis. Figure 2.15 shows our utilization results. With RC only, the utilization of Fixed PIM in Hetero PIM is improved by up to 66% (VGG-19); With OP, the utilization of Fixed PIM is further improved by up to 18% (AlexNet); With RC and OP, the utilization of Fixed PIM is close to 100%. The reason for the poor hardware utilization with neither RC nor OP is the lack of scheduling for the operations that do not have sufficient parallelism or cannot be completely offloaded to Fixed PIM.

2.6.6 Mixed Workloads Analysis

We also evaluate the case, when multiple models co-run in the same system [91]. We co-run two NN training models: a CNN model and a non-CNN model. The CNN model can

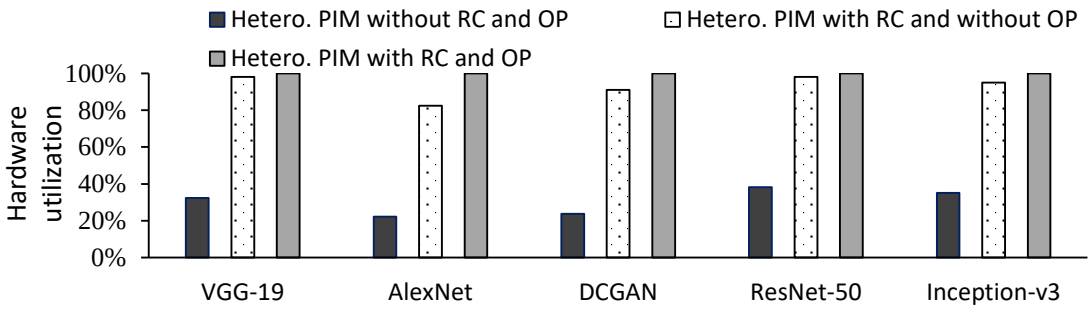


Figure 2.15. Hardware utilization with and without RC and OP.

execute on CPU and PIMs, subject to our runtime scheduling; the non-CNN model executes on CPU or the programmable PIM, when they are idle. Figure 2.16 shows the results of six co-run cases. In each case, "Hetero. PIM" indicates that we simultaneously execute both models, with the total execution time matched (i.e., when the CNN model executes for one step, the non-CNN model can execute one or multiple steps because the latency of the CNN model in one step can be longer than the non-CNN model in one step); "Sequential Execution" indicates that we execute the two models one after another in serial.

The results show that Hetero. PIM achieves 69%-83% performance improvement compared with Sequential Execution. Such improvement comes from high utilization of CPU and the programmable PIM in our design. With Sequential Execution, there can be no operations available to execute even though CPU and the programmable PIM are idle due to dependency between operations within the same model. Hetero. PIM avoids hardware idling, because operations across different models have no dependency and can execute simultaneously.

2.6.7 Energy Efficiency Analysis

We study energy efficiency of the PIMs with different frequencies as in Section2.6.4. We use energy-delay-product (EDP) as the metric to evaluate energy efficiency. Figure 2.17 (a) shows the results. The figure reveals that the most energy efficient point is not the original frequency for the five models. The 4× frequency is the most energy efficient for the five models. The tradeoff between energy consumption and execution time leads to such results. Thus, we conclude that

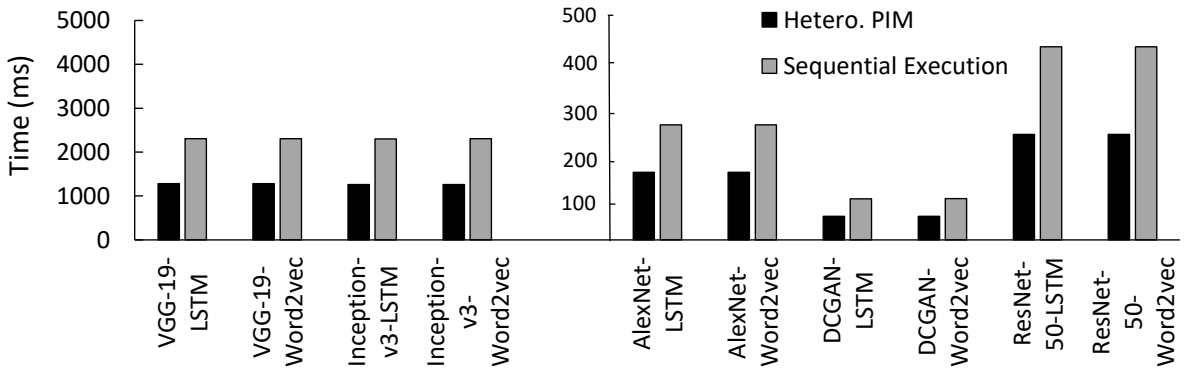


Figure 2.16. Execution time of multiple NN training models with our design and sequential execution, respectively.

higher frequency tends to be more energy efficient for NN model training. Figure 2.17 (b) compares power consumption between GPU and Hetero PIM with different frequencies. In general, GPU is very power hungry. It consumes 1.5× to 2.6× more power than Hetero PIM with high frequency (4×). Compared with GPU, Hetero PIM can be highly power efficient.

2.7 Related Work

Whereas previous PIM-based accelerator designs [26–28, 48, 55, 58, 94, 103] investigated the mapping of workloads on either fixed-function or programmable PIMs, it is unclear how to coordinate software and hardware designs to best utilize PIM technologies to support the heterogeneity requirement of NN training workloads.

Processing-in-memory for machine learning. Recent PIM-based machine learning accelerator designs strive to leverage the memory cells of nonvolatile memory technologies to execute NN inference operations [31, 48, 128, 141]. However, NN training typically incorporates substantial complex operations as we identified. It is difficult to accommodate these complex operations in previous processing-in-memory-cell designs. Azarkhish et al. [34] and Schuiki et al. [126] adopt RISC-V cores [8] and a streaming coprocessor in die-stacked DRAM to accelerate convolution networks or SGD. However, the RISC-V cores are merely used to control the arithmetic elements in the streaming coprocessor. Furthermore, both designs require users to

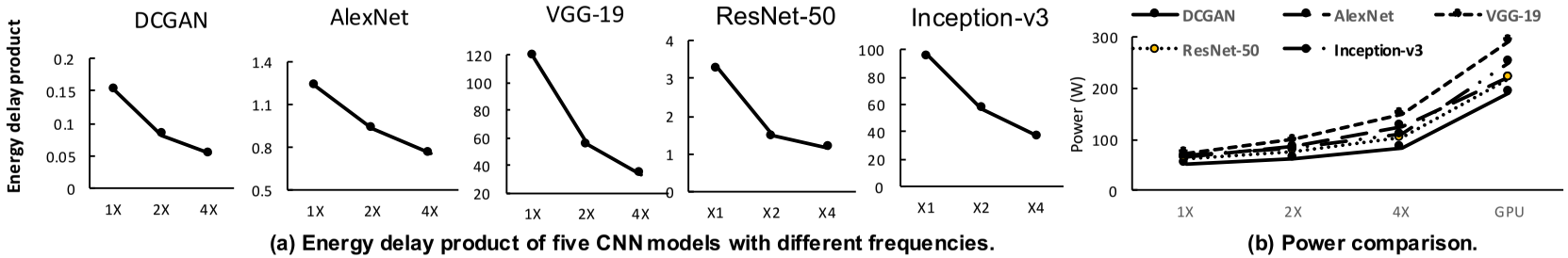


Figure 2.17. Energy efficiency and power with 3D memory frequency scaling.

modify code and perform tiling based on new APIs. Schuiki et al.'s study [126] only focuses on a specific operation (SGD). Azarkhish et al.'s design [34] primarily aims at inference and requires data to be carefully laid out in memory with 4D tiling. This constraint on data layout leads to inefficient training, because intermediate activations after each layer need to be re-tiled [126]. Neurocube [77] accelerates CNN inference and training by integrating programmable processing elements in the logic layer of 3D die-stacked DRAM. However, using programmable PIMs alone cannot provide the massive parallelism and execution efficiency enabled by heterogeneous PIMs. Furthermore, the aforementioned previous studies do not consider dynamic runtime scheduling of operations. Our experiment results demonstrate an efficient heterogeneous PIM design with runtime scheduling.

Processing-in-memory for general applications. Fujiki et al. [58] proposed a ReRAM-based in-memory processor architecture and data-parallel programming framework. The study introduces a compact instruction set for memory array with processors. The programming framework combines dataflow and vector processing, employs TensorFlow input, and generates code for in-memory processors. Our work also employs TensorFlow, but optimizes operations scheduling and introduces PIM heterogeneity. Ahn et al. [27] explores mapping of PIM operations based on data locality of applications, while we schedule operations in multiple dimensions – hardware utilization, data locality, and data dependency. Ahn et al. [26] introduced PIM for parallel graph processing. The design offers an efficient communication method between memory partitions and develops prefetchers customized for memory access patterns of graph processing. Other works introduce PIM architectures based on 3D-stacked memory. For example, Zhang et al. [146] presented an architecture for programmable, GPU-accelerated, in-memory processing implemented using 3D die-stacking. The throughput-oriented nature of GPU architectures allows efficient utilizaztion of high memory bandwidth provided by 3D-stacked memory, while offering the programmability required to support a broad range of applications. Akin et al. [28] presented a set of mechanisms that enable efficient data reorganization in memory using 3D-stacked DRAM. However, the aforementioned studies cannot efficiently accelerate

NN training workloads, because they cannot fully accommodate the heterogeneous computing requirement in NN training. Furthermore, these studies do not consider efficient programming model and runtime system to accommodate the hardware heterogeneity as explored in our study.

Other accelerator optimization methodologies for machine learning. Recent works explored software- and hardware-based approaches for a variety of inference acceleration [30, 46, 47, 56, 116]. Most of these works focused on improving performance and energy efficiency of NN inference. However, training is much more compute and memory intensive than inference. The data movement overhead in training is much more significant. Several prior studies [72, 119, 120] investigated architecture design for NN training. However, these studies focus on addressing the memory capacity constraint issues caused by a large amount of feature maps generated in CNN training. The data movement bottleneck is not fully explored.

2.8 Conclusion

In this chapter, we propose a software and hardware co-design of heterogeneous PIM approach, combining the power of programmable PIM and fixed-function PIMs, for NN training. Our software design enables (1) a portable and unified programming model across CPU, fixed-function PIMs, and programmable PIM; (2) runtime scheduling that effectively optimizes PIM hardware utilization and maximizes NN-operation-level parallelism. Our design not only allows natively training models to execute on heterogeneous PIM, but also enables easy maintenance of machine learning frameworks. Our design achieves significant improvement in performance and energy efficiency with various NN training workloads.

Chapter 2 contains material from ”Processing-in-Memory for Energy-efficient Neural Network Training: A Heterogeneous Approach” by Hengyu Zhao, Jiawen Liu, Matheus A. Ogleari, Dong Li, Jishen Zhao, which appears in International Symposium on Microarchitecture (Micro), Oct. 2018. The dissertation author was the primary investigator and author of this material.

Chapter 3

Safety-Aware Computing System Design in Autonomous Vehicles

3.1 Introduction

Today, autonomous vehicles (AVs) incorporate sophisticated suites of sensors, such as cameras, light detecting and ranging (LiDAR), and radar (Figure 3.1); These sensors are backed by advanced computing system software and hardware that interpret massive streams of data in real-time. As such, autonomous driving promises new levels of efficiency and takes driver fatigue and human errors out of the safety equation. However, the perceived technology will shift the burden of safety guarantee towards the vehicle system, and therefore, we need to ensure the safe operation of autonomous driving systems before releasing them to the public. As a consequence, the U.S. Department of Transportation recently released "A Vision for Safety" [105] to offer a path forward for the safe deployment of AVs and encourage ideas that deliver safer autonomous driving. Safety remains one of the most critical requirements for autonomous driving system design.

This chapter focuses on nominal safety (discussed in Section 3.2.2) of high automation vehicles, i.e., Level-4 autonomous vehicles. While traditional cars rely on human drivers to make driving decisions, future high automation vehicles will depend on the computing system for driving decision making (Section 3.2). Therefore, the computing system is one of the most critical components to ensure safety.

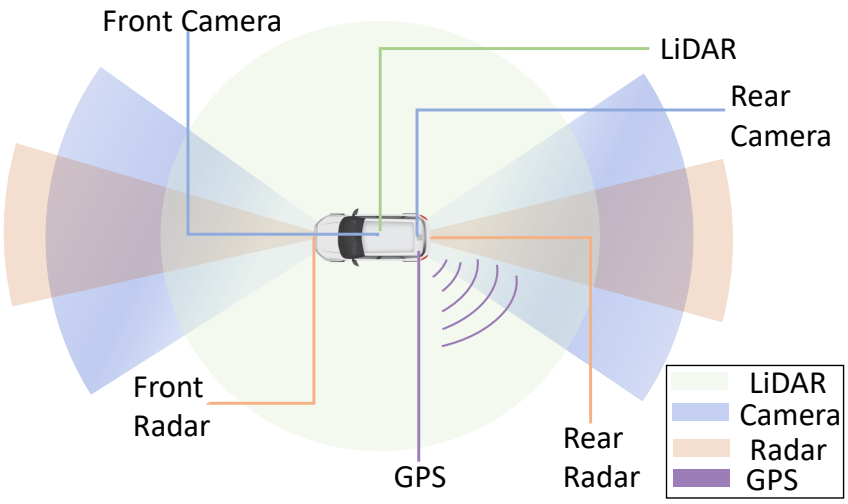


Figure 3.1. Top view of a typical autonomous vehicle with various sensors.

To investigate safety implications on the AV computing system design, we performed a field study (Section 3.3) by running Pony.ai's industrial Level-4 autonomous driving fleets under testing in various locations, road conditions, and traffic patterns. Based on the knowledge learned from extensive data collected during our field study, we identify that traditional computing system metrics, such as tail latency, average latency, maximum latency, and timeout, do not accurately reflect the safety requirement of AV computing system design (Section 3.3.2). Instead, the level of AV safety is a non-linear function of accumulative instantaneous computing system response time and various external factors – including the AV's acceleration, velocity, physical properties (e.g., braking distance), traffic, and road condition.

To guide safety-aware AV computing system design, we propose "safety score", a safety metric that measures the level of AV safety based on computing system response time and the aforementioned external factors in a non-linear format (Section 3.4). Our safety score is rigorously derived from the published industrial formal AV safety model, Responsibility-Sensitive Safety (RSS) safety model [129]. RSS focuses on general rules for an AV to keep a safe distance with surrounding vehicles. Our safety score bridges such a conceptual model to AV computing system design with a quantitative approach. We hope the proposed safety score will be an essential contribution to the existing computer system metric portfolio. To facilitate

the computation of safety scores, we also propose a latency model for offline computing system design evaluation. In summary, this chapter makes the following contributions:

- We provide a tutorial of state-of-the-art AV systems based on our survey and the practical experience on our AV systems. We offer a detailed discussion of automation levels, classification of safety requirements and standards, and computing system software and hardware organizations.
- We perform a field study with our AV fleets. We present observations and safety-aware computing system design challenges based on the extensive data collected from our field study.
- We propose the safety score, a metric that evaluates the level of safety in AV computing system design. By analyzing the safety score, we present a set of implications on safety-aware AV system design.
- We propose the perception latency model, which allows AV computing system architects to estimate the safety score of a given computing system and architecture design in various driving conditions, without physically testing them in a car.
- We demonstrate an example of utilizing our safety score and latency model to guide AV system computation resource management with a hardware and software co-design.

3.2 State-of-the-art AV System

High automation cars are under development and testing in both industry [1, 5, 10, 16] and academia [11, 14, 15]. To understand the current status of AV development, we examined the state-of-the-art AV design, AV safety, and hardware and software organizations.

3.2.1 Levels of Automation

The Society of Automotive Engineers (SAE) categorizes autonomous driving systems into six automation levels, which range from no automation (Level-0) to full automation (Level-5) [124]. Most commercialized autonomous driving systems are on partial automation (Level-2)

and conditional automation (Level-3). At Level-2, the vehicle can steer, accelerate, and brake in certain circumstances; however, the driver needs to respond to traffic signals, lane changes, and scan for hazards. For example, Tesla's latest Model S [5] is a Level-2 commercialized car. At Level-3, the car can manage most aspects of driving, including monitoring the environment; but the driver needs to be available to take over at any time when the vehicle encounters a scenario it cannot handle [9]. For example, NVIDIA recently announced a partnership with Audi to build Level-3 autonomous driving cars [1]. High automation (Level-4) AVs are under development. A Level-4, AV drives itself almost all the time without any human input, but might be programmed not to operate in unmapped areas or during severe weather. For instance, Waymo [10], originated as a project of Google, is developing a Level-4 autonomous driving system [97]. Full automation vehicles (Level-5), which can operate on any road and in any conditions that a human driver could manage, is not yet demonstrated. But this is a long-term goal of autonomous driving development.

This chapter focuses on Level-4 AVs, which introduce much more critical safety requirements on the computing system than lower-level AVs. For example, Level-3 AVs treat human drivers as a backup of the AV computing system; but Level-4 AVs entirely rely on the computing system to make driving decisions in almost all scenarios.

3.2.2 AV Safety

In automotive industry, safety requirements are classified into functional safety and nominal safety [129]. Functional safety refers to the integrity of operation of vehicle's electrical system, including hardware and software. For example, a hardware failure or a bug in software will both lead to a functional safety hazard. Functional safety of vehicles is standardized by ISO 26262 [69], which defines various automotive safety integrity levels that offer (a) failure-in-time targets for hardware and (b) systematic processes for software development and testing that conform with appropriate systems engineering practices.

However, unlike in a conventional vehicle, computing system plays an integral role in an

AV. Even functionally safe, AV can still crash due to untimely or unsafe driving decision. As Riccardo Mariani, Intel Fellow and chief functional safety technologist in the internet of things group, pointed out, current ISO standard of functional safety is inadequate to ensure the safety of AV [70]. This falls in the domain of nominal safety. Nominal safety refers to whether the AV makes timely and safe driving decisions, assuming that the hardware and software are operating error free (i.e., functionally safe).

This chapter focuses on nominal safety of AV. While software module algorithm design determines whether a driving decision (e.g., braking, acceleration, or steering) is safe, timely driving decision making heavily relies on the performance of computing system and architecture design. Two recent studies by Mobileye [129] and NVIDIA [106] defined formal nominal safety rules for AVs. However, these studies focus on how to make planning and control decisions to keep a safe distance with surrounding vehicles. To our knowledge, no previous research investigates how to ensure timely driving decision making with safety-aware computing systems design. As such, this chapter focuses on exploring safety-aware computing system design methodology that achieves timely driving decision making, by mapping formal safety requirements onto AV computing system design.

Note that it is impossible to guarantee that an AV will never be involved in an accident [129]. It is also difficult to predict or control the activity of other vehicles [106]. Therefore, we intend to guarantee that the AV will be sufficiently careful to avoid becoming a cause of an accident, the same as recent AV safety studies [129].

3.2.3 AV Computing System Architecture

The AV computing system performs similar tasks as a driver of a conventional car, including localization, perception, and planning and control tasks (described in Section 3.2.4). In the rest of this chapter, we call AV computing system as "AV system" for short.

While embedded or low-end processors may be sufficient to meet the computation requirement of Level-2 and Level-3 AVs, current Level-4 AV system designs typically adopt high-

Table 3.1. Computing system and architecture configurations.

CPU	Intel Xeon processor
Main memory	16GB DDR4
Operating system	Ubuntu
GPU	NVIDIA Volta architecture
Device memory	12GB HBM2

end heterogeneous architectures, which comprise sophisticated CPUs, GPUs, storage devices (e.g., terabytes of SSDs), FPGAs [12] and other accelerators, to accommodate the intensive computation demand. As such, high automation AV systems are fundamentally different from conventional real-time embedded systems [151, 152, 154]. Table 3.1 lists the heterogeneous architecture employed by our prototypes.

Implication: Although the computing system adopts server-grade processors, it is impossible to minimize the latency of all computation tasks due to the vast amount of interdependently executing software modules (discussed in Section 3.2.5) and the finite hardware resources in the system. As such, it is critical to effectively utilize the limited hardware resources, while meeting given safety requirement.

3.2.4 Tasks of AV Computing System

The computing system interacts with various sensors and the car control system to perform three primary functions: localization, perception, and planning and control. Figure 3.2 illustrates the general working flow of the state-of-the-art Level-4 AV computing system.

Localization. Localization identifies the location of the vehicle on a high definition map, based on the information obtained from LiDAR, camera, and GPS. The high definition map stores static information on the way, such as lane lines, trees, guardrails, and the location of traffic lights and stop signs [43]. GPS offers a prediction of the current location of the vehicle on a global scale; but such prediction is not sufficiently accurate to allow the computing system to perform driving tasks, e.g., staying within a lane on the road [44]. Therefore, GPS information needs to be accompanied with LiDAR and camera "perception" (described below) of surrounding

static objects, to interpret the local environment accurately.

Perception. Perception detects and interprets surrounding static (e.g., lane lines, trees, and traffic lights) and moving (e.g. other vehicles and pedestrians) objects with three types of sensors, LiDAR, cameras, and millimeter-wave radar, as shown in Figure 3.1. In the rest of the chapter, we refer all the objects, which can be detected by perception, i.e., in the detection range, as "obstacles". Localization typically only requires infrequent perception of static objects, while traffic interpretation requires frequent (e.g., per-frame – each frame of data collected by the sensors) perception of both static and moving objects.

Compared to camera and radar, LiDAR offers much higher accuracy on detecting the position, 3D geometry, speed, and moving direction of obstacles [67, 125]. Therefore, most of current Level-4 AVs highly rely on LiDAR to perform the perception task [53, 64, 75]. LiDAR continues to emit lasers across a 360 degree view. It receives reflected signals, whenever an obstacle blocks the laser. Based on the received signals, we can build a "point cloud" to represent the location and 3D geometry of the obstacle [122]; each point in the graph represents a received signal.

In most cases, the computing systems can identify obstacle properties based on the point clouds generated from LiDAR data, as shown in the left part of Figure 3.5. Cameras facilitate the obstacle interpretation by capturing color information (the right part of Figure 3.5). Radar is used to detect the radial speeds of moving obstacles [88]. After sensor data collection, the computing system performs a "perception fusion" to integrate the data from the three types of sensors.

Overall, perception obtains accurate information of location, geometry, speed, and moving direction of obstacles. Such information is essential for localization and making appropriate driving decisions to avoid collisions in planning/control.

Planning and control. The essential functions of planning and control include prediction, trajectory planning, and vehicle control. Their dependency relationship is shown in Figure 3.2.

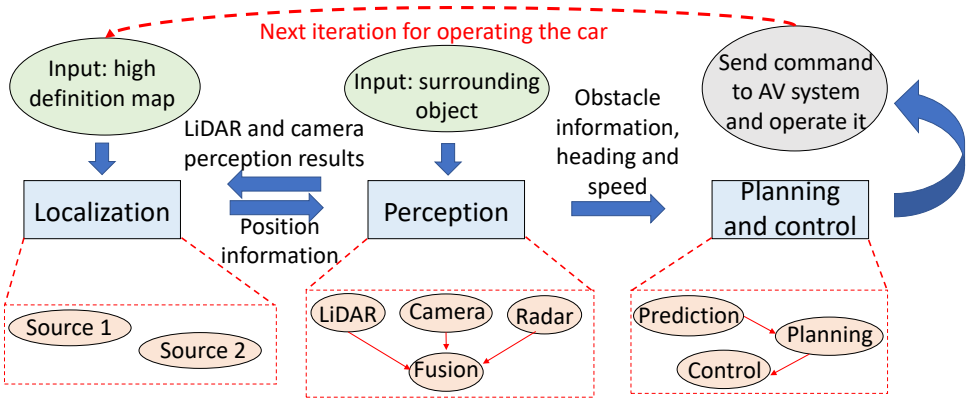


Figure 3.2. AV computing system tasks.

A motion planner employs localization and perception information to make driving control decisions, such as steering angle, acceleration, and braking of the vehicle. Functional safety is of critical importance to the motion planner. As such, a backup mechanism, commonly radar or sonar [93], is used to verify the conclusion informed by other components of the system. With the perception results, the prediction function tracks the behavior of obstacles at real-time and predicts their next movements. The computing system generates an optimal trajectory for the vehicle, based on the prediction and perception results. The control function will control the steering, acceleration, and deceleration of the vehicle based on the driving decision made by the planning function.

3.2.5 LiDAR Perception

As perception in most of the current Level-4 AV designs heavily relies on LiDAR, we present more details of LiDAR perception.

Dependency graph. To process LiDAR data, the computing system executes a considerable number of inter-dependent software modules (i.e., software programs) at real-time. To understand the dependency relationship among these software modules, we build a dependency graph based on our AV system. Figure 3.3 shows an example[1]. LiDAR perception consists of

[1]We only show renamed high-level modules due to confidential reasons.

a set of main modules (shown in blue) and sub-modules (shown in orange). For example, the *Labeling* module first sorts out a rectangular region-of-interest (ROI), which restricts the range of LiDAR detection; then, it calls its four sub-modules to perform finer-grained labeling within the ROI. As another example, *Segmentation* extracts useful semantic and instance information (e.g., surrounding vehicles and pedestrians) from the background. The *Merge* module clusters the points, which are likely to belong to the same object, in a point cloud; the *Filter* modules process the point clouds. Some software modules, e.g., *Segmentation*, do not only impose long latency, but also have a strong dependency with other software modules.

Implication: Due to their long latency and high dependency, certain software modules contribute more to the overall perception latency and computing system response time than other modules. Therefore, these modules are typically more safety critical.

3.2.6 Types of Computing System Workloads

AV system workloads consist of both machine learning and non-machine-learning (non-ML) software modules. In our AVs, camera perception software modules are commonly deep-learning-based computer vision workloads; but most of the software programs in our AV system are non-ML. For example, some classification and segmentation modules can employ Support Vector Machine (SVM) [127] and fully connected networks; camera data perception may adopt ResNet [63]. Currently, lots of the software programs running in our AV system, even for the perception task, are non-ML workloads; many of these non-ML workloads are as time-consuming and compute-intensive as deep learning workloads. One such example is the *Segmentation* module shown in Figure 3.3.

Furthermore, many software modules can execute on both CPU and GPU with the same algorithms but different implementations. This introduces substantial flexibility in managing and allocating the various hardware resources among the software workloads.

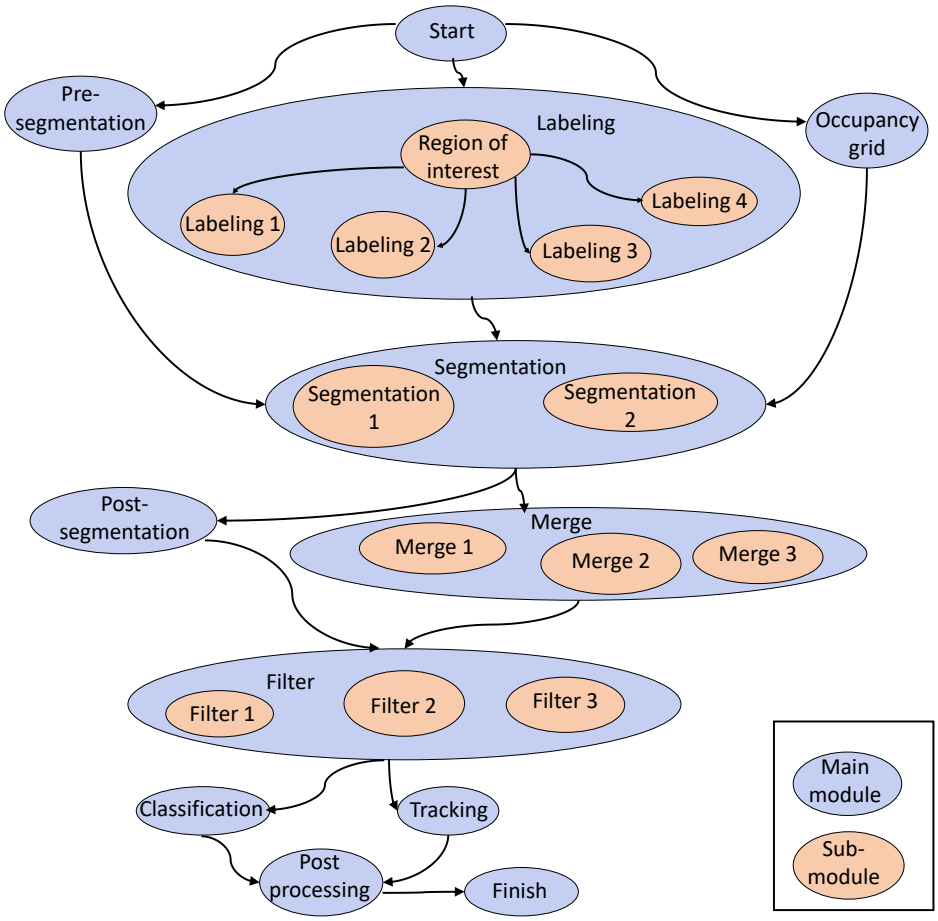


Figure 3.3. Dependency graph of LiDAR perception.

3.3 Field Study and Observations

To explore realistic AV safety requirement and computing system hardware/software behaviors, we run a fleet of Level-4 AVs in various locations, road conditions, and traffic patterns over three contiguous months. Our field study yields over 200 hours and 2000 miles of traces with a considerable size of data. The data is collected at the granularity of one sensor frame, which is the smallest granularity that the computing system interprets and responses to obstacles. With each frame, we collect (1) the execution latency of each software module, (2) total computing system response time, (3) the utilization of computing system hardware resources (CPU, GPU, and memories), (4) the environment information obtained by the sensors (e.g., the distribution of static and moving obstacles), and (5) instantaneous velocity and acceleration of the AV.

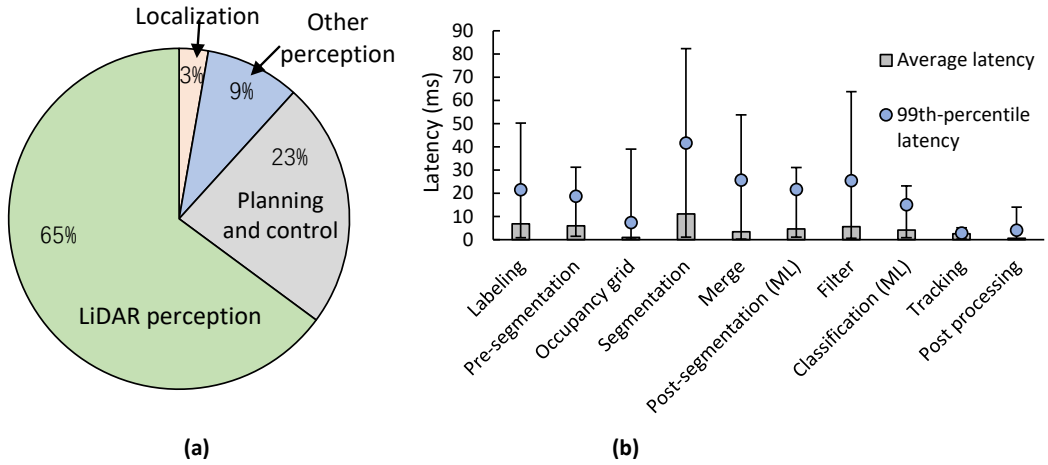


Figure 3.4. (a) Computing system response time breakdown. (b) The latency of LiDAR perception main modules.

Our field study is performed as part of the company's routine road tests. The data we collected incorporates a large variety of scenarios, including near-accident scenes.

3.3.1 Observations

Breakdown of computing system response time and the impact of architecture design. As shown in Figure 3.4(a), the majority (74%) of computing system response time is consumed by perception. LiDAR perception is the major contributor to perception latency. As nominal safety requires a timely response, it is critical to ensure timely LiDAR perception. In addition, we observe that most of the workloads running in our AV system are compute-intensive. Various software modules have a large variety of computation requirements and parallelism characteristics. As a result, native heterogeneous system hardware resource management easily results in unbalanced compute resource utilization, leading to significant delays in software execution.

Safety-critical software module analysis. We further analyze the execution latency of the main modules. Figure 3.4(b) shows an example with LiDAR perception modules by illustrating average, maximum and minimum, and tail latency of the ten main software modules across thousands of executions during our months of field study. We have two critical observations.

First, maximum, average, and tail latency yields different criticality rankings on the modules. For example, specific modules with high maximum latency (e.g., *Occupancy grid*) have low average and tail latency. We observe that maximum latency is typically achieved in safety-challenging scenarios, such as heavy traffic and near-accident situations, e.g., Figure 3.5 explained below. Therefore, maximum latency better indicates safety-criticality of a software module than other formats of latency. Second, we observe that safety-critical modules also typically have strict inter-dependency with other modules in the dependency graph. Examples include *Segmentation* and *Filter*. Based on the two observations, safety-critical modules can be identified as those having long maximum latency and strict inter-dependency with other modules.

Timely traffic perception is the most critical to AV safety. Due to the limited ranges of sensor detection, the computing system has limited time (typically within one or several sensor sampling intervals) to interpret an obstacle in the traffic after it enters the sensor-detectable region. Moreover, as perception consumes the majority of computing system response time, timely traffic perception is the most critical to AV safety. Specifically, LiDAR perception is the most safety critical in our AV system (and many other Level-4 AV implementations), because most of the perception task relies on LiDAR. In fact, we encountered several safety emergencies, when the AV needed to perform a hardbrake due to the delay of LiDAR perception. Figure 3.5 illustrates one of such safety emergency incidents we encountered. As shown in the right camera view, a white SUV is passing through an intersection, when our AV intends to enter the same intersection[2]. Our system recognized the traversing SUV within a LiDAR sampling interval and ended up performing a hardbrake as shown in Figure 3.5(d). But if the perception had taken slightly longer time, a collision would have happened.

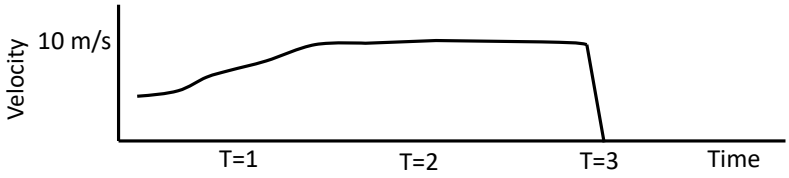
Correlation between perception latency and obstacle distribution. We observe that perception latency depends on both computing system configuration and the distribution of obstacles (especially in traffic). With a given computing system hardware configuration, obstacles

[2]The white SUV entered the intersection before the traffic light turned red, passing through at a high speed.

(a) T=1, (Left: LiDAR point cloud graph, Right: camera graph)

(b) T=2, (Left: LiDAR point cloud graph, Right: camera graph)

(c) T=3, (Left: LiDAR point cloud graph, Right: camera graph)



(d) Velocity-time graph.

Figure 3.5. An emergency hardbrake case that we encountered.

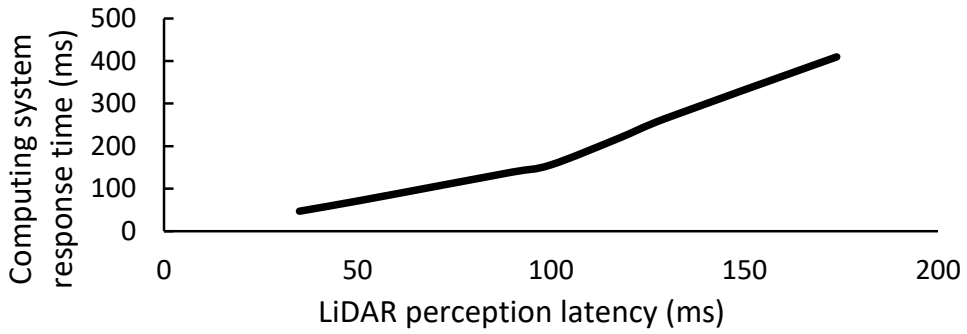


Figure 3.6. The relationship between LiDAR perception latency and computing system response time.

closer to the vehicle have a higher impact on perception latency than distant ones. Furthermore, obstacles that horizontally distribute in front of the vehicle have a higher impact on perception latency than those distributed in other directions. Finally, among the close obstacles, the denser the obstacle distribution, the longer the perception latency would be. The rationale underlying is that close, horizontally-distributed, and dense obstacles reflect more LiDAR signals than other obstacles, hence generate more data for LiDAR perception.

Latency accumulation effect. We observe a latency accumulation effect of certain software modules. Figure 3.6 illustrates an example of LiDAR perception modules: once the LiDAR perception latency exceeds 100ms, the system response time increases at a much higher rate (in the region with a steeper slope in Figure 3.6) with the increase of the LiDAR perception latency. The reason is two-fold. First, because the LiDAR has a fixed sampling rate (with a 100ms interval in our AV), the perception, if not completed within the sampling interval, will further delay the processing of subsequent frames. Second, as software modules are inter-dependent, a delay of one software module will further delay its dependent software modules. The slopes and thresholds of the accumulation effect vary across different software modules, depending on individual characteristics and the inter-dependency relationship with other modules.

Impact of other external factors. Besides traffic, several other external factors also have a critical impact on the safety requirement of AV system design. These factors include the acceleration and velocity of the AV and vehicles around it, AV's physical properties (e.g.,

braking distance), and road condition. Taking the scenario in Figure 3.5 as an example, if our AV had a higher velocity or longer braking distance, the collision would also have happened.

3.3.2 Safety-aware Design Challenges

As discussed in Section 3.2.2, this chapter focuses on nominal safety that ensures timely driving decision making by the computing system. It seems that timely driving decision making can be achieved by minimizing the computing system response time. However, this is challenging. To ensure achievable minimum response time within finite computing system hardware resources, performance optimization needs to be guided by specific metrics. Based on our field study, traditional computing system performance metrics, such as average latency, maximum latency, tail latency, and timeouts, do not accurately reflect the safety requirement of AV systems due to the following reasons.

First, AV safety is determined by instantaneous response time (Figure 3.5), instead of statistical patterns adopted in traditional performance metrics. Second, it is challenging to predict the response time due to the complex inter-dependent relationships between software modules and the latency accumulation effect. Third, AV safety is also determined by various external factors as discussed in our field study observations. As a result, the level of AV safety is not a simple weighted sum of the elements, but a non-linear function of response time as we identified in our study (Section 3.4). Finally, although timeout is widely used to measure real-time system performance [154], it is impractical to simply use timeout to evaluate the safety of AV system design: It is difficult to determine the threshold required to calculate the timeout of AV system response time, because each software module has a different threshold (e.g., different sampling interval with various sensors); External factors further lead to dynamically changing thresholds. Therefore, we need to rethink the fundamental metric used to guide safety-aware AV system design.

3.4 Safety Score

In this section, we propose the safety score, a metric that measures the level of AV safety based on computing system response time and external factors to guide safety-aware AV system design.

3.4.1 Safety Score Description

Our safety score is rigorously derived from the published industrial formal AV safety model – Responsibility-Sensitive Safety (RSS) [129]. RSS is developed by Mobileye, an Intel subsidiary that develops advanced driver-assistance systems (ADAS) to provide warnings for collision prevention and mitigation. RSS defines safe distances around an AV, formal mathematical models that identify the emergency moment, when the safe distances are compromised, and proper planning and control decisions an AV needs to perform during an emergency. RSS has been endorsed by various AV projects, such as Valeo [17] and Baidu's Apollo Program [13].

Unfortunately, RSS does not provide many insights on safety-aware computing system design. To give a quantitative guideline for safety-aware computing system design, we map RSS to the safety score based on our field study observations. The safety score is defined as the following equation (check Section 3.5 for precise mathematical formulation of safety score):

$$\text{Safety Score} = \begin{cases} \sigma[\alpha(\theta^2 - t^2) + \beta(\theta - t)], \ if\ t < \theta \\ \eta[\alpha(\theta^2 - t^2) + \beta(\theta - t)], \ elsewhere \end{cases} \tag{3.1}$$

The variables in the equation are described in Table 3.2. Here, t is the "instantaneous" computing system response time, which we define as the response time to one frame of sensor data (or several frames depending on the perception granularity of an AV implementation). We calculate t by the following equation:

$$t = \sum_{i=1}^{N} w_i(t_i) \tag{3.2}$$

Table 3.2. Variables in safety score.

Variable	Description
t	Instantaneous computing system response time, defined by Equation 3.2.
θ	Response time window.
α, β	Indicating the velocity and acceleration of an AV and surrounding vehicles, defined by Equation 3.4 in Appendix 3.5.
σ, η	Reward or penalty on the level of safety, when t is lower or higher than θ, respectively.

Here, t_i is the instantaneous latency of a safety-critical module on the critical path; N is the total number of such modules. Safety critical modules are determined based on the inter-dependency relationship in the dependency graph and the maximum latency via the extensive pre-product road tests (Section 3.3). $w_i()$ is an accumulation effect function, which reflects the contribution of t_i to system response time t. For each safe-critical module, we obtain a latency accumulation effect curve that can be described by a function $w_i()$. Figure 3.6 shows an example of such curves, where the x-axis is t_i and y-axis is the value of $w_i(t_i)$, but curve shapes vary across different modules.

The rest variables in Equation 3.1 reflect the impact of external factors. The response time window θ is the minimum time that the AV would take to avoid hitting another vehicle that is certain distance d away, if both vehicles keep moving at the same direction, velocity, and acceleration, i.e., neither of vehicles perform any response to a reducing distance. We allow AV developers to define the distance d in various manners, e.g., as (1) a minimum safe distance allowed by traffic rules, which is adopted by our experiments as an example to make our evaluation more concrete or (2) a user-defined confident safe distance. Figure 3.8 in Section 3.5 provides a more detailed analysis of θ.

Variables α and β reflect the velocity and acceleration of the AV and the surrounding vehicle, calculated by Equation 3.4 in Section 3.5. Safety reward σ and penalty η are user-defined coefficients, which indicate the user's confidence of safety. We allow users to adjust the reward

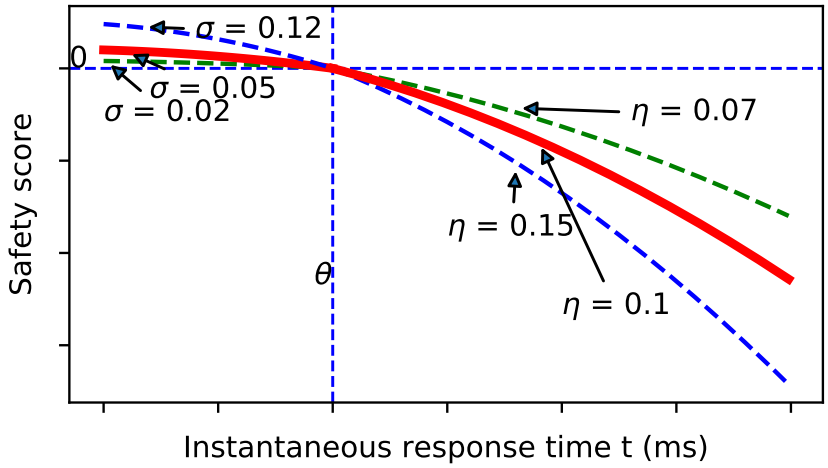


Figure 3.7. The non-linear relationship between safety score and instantaneous response time.

and penalty based on their own road test experiences, such as the velocity of the vehicles in the traffic that the AV is likely to run into, road condition, and AV braking distance. We discuss more detailed implications of the reward and penalty in Section 3.4.2. Section 3.5 provides more detailed discussion of these variables.

No clear boundary is defined between safe and unsafe in various AV safety models [106, 129]. As such, the safety score does not intend to give a threshold of safe versus unsafe either. Instead, it implies the level of safety – higher safety score (could be either positive or negative) indicates safer AV system design.

3.4.2 Implications to Computing System Design

Based on our practical experience, safety score provides the following key implications on the safety-aware AV system design.

Nonlinearity. The safety score is a non-linear function of instantaneous response time t. Safety-aware computing system design approach highly relies on the shape of the curve. Figure 3.7 lists a few examples of safety score curves as a function of t. With a given set of external factors, the shape of the curve is determined by response time window θ, safety reward σ, and safety penalty η. Based on our road tests, the majority of safety emergency scenarios

happen when $t > \theta$. When $t < \theta$, further optimizing the instantaneous response time does not significantly improve the level of safety. Therefore, in practice the shape of the safety score curve appears like the bold line (σ=0.05, η=0.1) shown in Figure 3.7, with a lower reward than penalty. As such, computing system design needs to prioritize reducing the instantaneous response time t, when $t > \theta$.

Instantaneous latency. As we discussed before, the safety score is computed by instantaneous response time per sensor frame. Therefore, it is infeasible to adopt traditional performance metrics calculated based on statistical performance behaviors, such as tail, average, and maximum and minimum latency. Section 3.8.1 quantitatively compares various metrics.

Safety impact across various software modules. Due to the complex inter-dependency relationship and latency accumulation effect of the software modules, the instantaneous latency of different software module has different impact on safety score. This is modeled by Equation 3.2, where each software module has a disparate form of the contribution to the instantaneous AV system response time.

3.5 Safety Score Formulation

Responsibility-Sensitive Safety (RSS) [129] formalizes safety as assuring five common-sense rules, which AV driving policy should obey. Specifically, the policy will choose safe actions under various circumstances. The action is safe, if (1) given the current driving state and the actions and (2) when the surrounding vehicles are acting adversely but reasonably, a collision can be avoided.

More formally, a given driving state $\{v, a_{max,+}, a_{min,-}\}$ of the AV and the driving state $\{v', a'_{max,+}, a'_{min,-}, a'_{max,-}\}$ of the vehicle that might have collision with the AV (referred to as "obstacle-of-attention"), together determine the minimum distance d_{min} to avoid collision by Equation 3.3:

$$d_{min} = vt + \frac{1}{2}a_{max,+}t^2 + \frac{(v + ta_{max,+})^2}{2a_{min,-}} + m(v't' + \frac{1}{2}a'_{max,+}t'^2) + n\frac{(v' + t'a'_{max,+})^2}{2a'_{*,-}} + d_\mu \tag{3.3}$$

Here, v and v' are velocities of the AV and the obstacle-of-attention, respectively; $a_{min/max,+/-}$ is the minimum/maximum acceleration of speeding up(+) or slowing down(−). Let t and t' be the response times of the AV and the obstacle-of-attention. All variables are non-negative: velocities and accelerations only represent the absolute value.

Parameters m and n are used to represent whether the two vehicles are driving in the opposite or the same directions. The values are scenario-dependent. For example, $(m = 1, n = 1)$ indicates that the two vehicles, the AV and the obstacle-of-attention, are driving towards each other, either along the lane direction or perpendicular to the lane direction. The reasonable conditions are the obstacle-of-attention takes some response time t' before it applies a brake; the most adverse condition is to use the minimum slowing down acceleration $a'_{*,-} = a'_{min,-}$. When the two vehicles are driving towards the same direction, which is represented by $(m = 0, n = -1)$, the most adverse condition is that the obstacle-of-attention instantly brakes ($t' = 0$) with highest possible acceleration $a'_{*,-} = a'_{max,-}$.

Parameters $t', a'_{*,+/-}, v', a_{*,+/-}, d_\mu$ depend on various environmental conditions and user's safety preference. In practice, under bad weather or road conditions, we may adopt lower v' and $a'_{max,+/-}$ than normal conditions, because vehicles tend to drive slower and apply cautious speed changes. d_μ is the minimum distance between the two vehicles, when they perform safe actions [129] and full stop. Users may specify larger d_μ to allow a larger safety margin or smaller d_μ to achieve higher driving flexibility. Higher $a_{*.+/-}$ enables the AV to react more agile, but may lead to the worse passenger experience. At any time, the AV speed

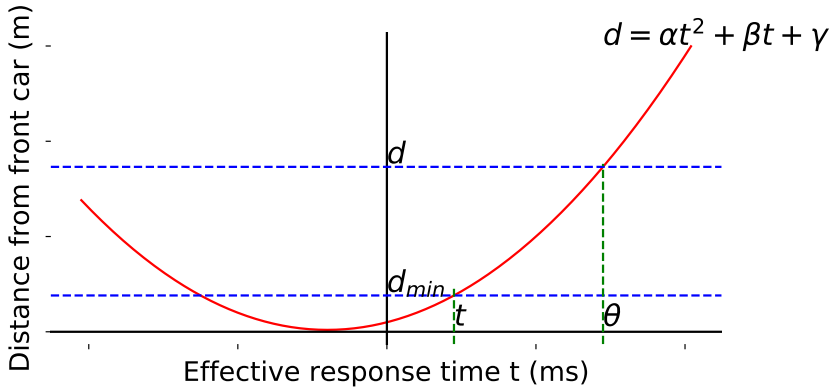


Figure 3.8. Illustration of d-θ and t-d_{min}.

v is known. The AV sensors detect the speed of the obstacle-of-attention. Collapsing all the parameters and grouping by the only variable t gives a quadratic form as Equation 3.4. Notice that α and β are non-negative.

$$
\begin{aligned}
&d_{min} = \alpha t^2 + \beta t + \gamma, \\
&where\ \alpha = \frac{1}{2} a_{max,+} + \frac{a_{max,+}^2}{2a_{min,-}},\ \beta = v + \frac{v a_{max,+}}{a_{min,-}},\ \gamma = \frac{v^2}{2a_{min,-}}
\end{aligned}
\tag{3.4}
$$

According to RSS [129], if the current distance d is larger than d_{min}, the safety criteria is met. Without loss of generosity, a safety score function can be defined as Equation 3.5. σ is the reward specified by users to achieve extra margin from minimal safety distance; η is the penalty of unit distance less than the minimum safety distance.

$$
\text{Safety Score} = \begin{cases} \sigma(d - d_{min}), if\ d > d_{min} \\ \eta(d - d_{min}), elsewhere \end{cases}
\tag{3.5}
$$

We employ Equation 3.4 to bridge the safety score with computing system response time. First, the current distance d between the AV and the obstacle-of-attention will determine the response time window θ that will avoid the collision. The response time window θ is the time

that the AV would take to hit the obstacle-of-attention, if both vehicles keep moving at the same velocity and acceleration – or more formally, if both vehicles conform the Duty of Care, i.e. an individual should exercise ”reasonable care” while performing actions that might harm the safety of others [129]. For instance, the Duty of Care allows the front car to perform break, such that the distance to the rear car is reduced, but prohibits the front car to backup, when followed by another car. We determine θ by solving the following function:

$$d = \alpha\theta^2 + \beta\theta + \gamma \tag{3.6}$$

$$\text{Safety Score} = \begin{cases} \sigma[\alpha(\theta^2 - t^2) + \beta(\theta - t)], if\, \theta > t \\ \eta[\alpha(\theta^2 - t^2) + \beta(\theta - t)], elsewhere \end{cases} \tag{3.7}$$

Let t be the AV computing system response time and determine $d_{min} = \alpha t^2 + \beta t + \gamma$. Figure 3.8 depicts the relationship between Equation 3.4 with the current distance d and instantaneous computing system response time t. Then the safety score function can be written as Equation 3.7.

3.6 Latency Model

The goal of our latency model is to estimate the instantaneous response time in safety score, given obstacle distribution and computation resource allocation. Our field study shows that perception latency is a significant contributor to system response time (Section 3.3). Therefore, we focus on modeling perception latency.

Our model is built based on two observations: first, closer and denser obstacles lead to higher perception latency (Section 3.3); second, latency also depends on computation resources allocated to run a software module[3]. Figure 3.9 shows an overview of our latency model. The

[3]In our current AV system, such hardware resource allocation refers to whether executing a software module on CPU or GPU. Although our system does not adopt FPGA or other accelerators, our latency model applies to general heterogeneous architecture design with a variety of hardware resources

model input includes (1) obstacle density distribution and (2) computation resource allocation of each software module. The output is the estimated latency of each software module, i.e., t_i in Equation 3.2. Our latency model estimates t_i in two steps:

- We first estimate a baseline latency (τ_i), which is the latency of executing a software module on a baseline computation resource (e.g., CPU in our AV system), using a baseline latency model presented by Equation 3.8.
- Then, we calculate t_i with latency conversion ratios based on given resource allocation plan.

In the following, we use LiDAR perception as an example to discuss our latency model in detail and can surely apply it to other perception modules, such as cameras, radars, etc. Section 3.8.2 evaluates the accuracy of our latency model.

3.6.1 Model Input

Obstacle density distribution vector. We design an obstacle count map to represent obstacle density distribution. The map divides the sensor ROI into regular grids, with each grid storing the number of obstacles in it. For example, our AV has a $64m \times 50m$ ROI with a $2m \times 2m$ grid size. This grid size captures most moving objects on the road, e.g., vehicles and bicycles, but large obstacles are counted multiple times, as they spread across multiple grids. To express more accurate obstacle density, we adopt a hierarchy of three levels of obstacle count maps with a grid size of $2m \times 2m$, $8m \times 10m$, and $64m \times 50m$ (the whole ROI), respectively. With the map hierarchy, we generate an obstacle density distribution vector $\vec{x}$ as illustrated in Figure 3.9(b).

Computation resource allocation vector. We represent computation resource allocation by a resource index vector $\vec{r}$, where each element in the vector indicates the resource allocation of a software module. For example, our current AV implementation only has two resource allocation options, running on CPU (index=0) or GPU (index=1). Then, $\vec{r} = [0, 0, 1, ..., 0]$ indicates that software modules are executed on CPU, CPU, GPU, ..., CPU.

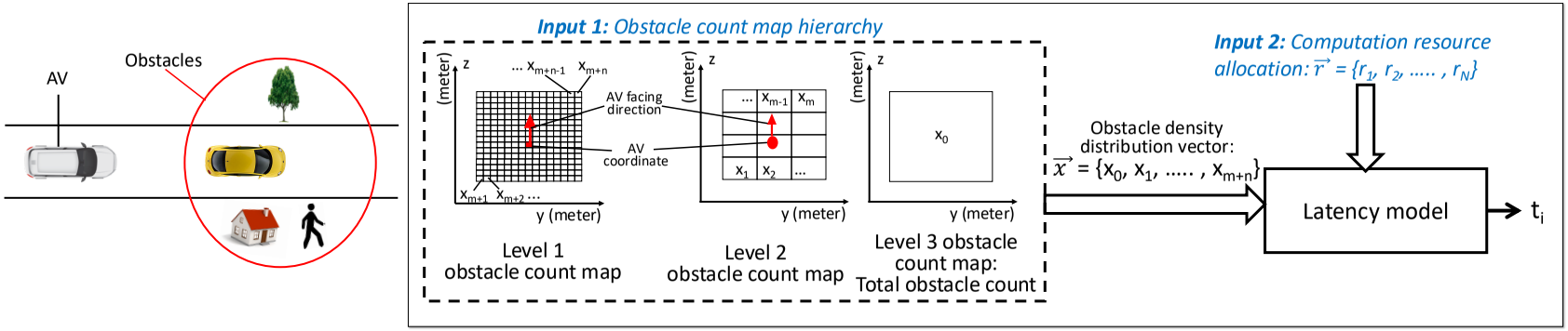


Figure 3.9. Overview of the proposed perception latency model. (a) Sources of perception latency. (b) Latency model working flow.

3.6.2 Latency Model

Perception software modules vary in complexity. We take into account the diversity in the algorithm complexity and model the baseline latency of each software module (τ_i) by the following equation,

$$\tau_i = \vec{a} \cdot (\vec{x} \odot \vec{x}) + \vec{b} \cdot (\vec{x} \odot log(\vec{x})) + \vec{c} \cdot \vec{x} + \vec{d} \cdot log(\vec{x}) + e \tag{3.8}$$

Here, x is a $m+n+1$ dimension obstacle density distribution vector, where m and n are the grid counts in the finest and second-level hierarchy, respectively. The operator $\odot$ is coefficient-wise vector product, and $\cdot$ is the inner vector product.

This is a standard curve fitting problem. To solve for coefficients $\vec{a}$, $\vec{b}$, $\vec{c}$, $\vec{d}$, and e in the equation, we perform linear regression on massive road testing data, which is used to generate the model input data, i.e., obstacle density distribution and corresponding baseline latency.

The final latency of each perception module then can be estimated by

$$t_i = \tau_i \nu(r_i) \tag{3.9}$$

Here, $\nu(r_i)$ is the ratio of executing the same module on a different computation resource than the baseline. This ratio is determined by exhaustive search of running a software module with various computation resource allocation configurations.

3.7 AV System Resource Management

We demonstrate an example of utilizing our safety score and latency model to guide the computing system design, by designing a heterogeneous computation resource management scheme. The goal of resource management is to determine the computation hardware resource allocation of each software module and the priority of software module execution to optimize safety score.

Overview. Our resource management has two phases, resource planning and resource scheduling. During the *planning phase*, we perform an exhaustive search of computation resource allocation and priority options of executing software modules in various obstacle distributions, to determine the options that maximize safety score given each obstacle distribution. This yields a resource management plan for each obstacle distribution. During the *scheduling phase*, we match the current obstacle distribution with resource management plans and determine the plan to use. To ensure sufficient resources to be scheduled for each software module, our AV system maintains redundant computation resources. Due to the large searching space, resource planning usually takes a long time. Therefore, we perform the planning phase offline, while the AV system only performs online scheduling. To further accelerate the online scheduling, offline planning groups obstacle distributions associated with the same resource management plan to clusters. As such, the scheduling phase only needs to match clusters.

3.7.1 Offline Planning

The offline planning phase analyzes the massive data collected by road tests to create a set of resource management plans with various obstacle distributions. A resource management plan includes (i) the computation resources allocated for software module execution (CPU or GPU in our current AV system design) and (ii) the priority of software modules, which is used to determine which module gets executed first, when multiple choices exist.

Algorithm 1 illustrates the offline planning procedure. For each field study sample, which comprises instantaneous computing system response time and the latency of each software module for an obstacle distribution, we perform an exhaustive search in the space of resource management plans. We categorize the samples, which achieve the highest safety score for the same resource management plan, in the same cluster. Based on our field study, samples in the same cluster have similar (1) obstacle density distribution and (2) perception timeout pattern, which is the number of continuous incidents where the perception latency exceeds the sensor sampling interval (e.g., a LiDAR sampling interval is 100ms). Therefore, we concatenate these

two as the feature, and compute the feature vector of each cluster by averaging the feature of each sample in the cluster.

This phase only need to be done once, whenever the AV adapts to a new location or physical system. In practice, we found that the clustering and the associated resource management plans are stable over time, even though users can also periodically run the offline phase to re-calibrate.

Algorithm 1. Offline planning.

Input : *Obstacle density distribution vector* $S \in \mathbb{R}^{N \times m}$
Input : *Timeout pattern* $K \in \mathbb{R}^{N \times m}$
Input : *Dependency graph* G
Input : *Computation resource management plans* P
Input : *Safety score function* L
Output : *Category features* $F \in \mathbb{R}^{k \times 2m}$
Init : *Category Dictionary* D
for $i = 1 \rightarrow N$ **do**
    $P_i^* = \text{argmax}_{P_j} L(G, P_j, S_i)$
    append $S_i \oplus K_i$ *to* $D[P_i^*]$
for $j = 1 \rightarrow k$ **do**
    for $q = 1 \rightarrow m$ **do**
        $F_{j,q} \leftarrow mean(D[P_j]_q)$

3.7.2 Online Monitoring and Scheduling

The online phase adopts a software/hardware co-design approach to (1) monitor the computing system execution and (2) match the resource management plans. Figure 3.10 shows a system design overview. When the AV is driving, the computing system continuously (1) monitors instantaneous system response time and the pattern of perception timeout and (ii) determines when and which resource management plan to adopt.

A loop of three steps implements this: (Step-1) perception timeout pattern monitoring, (Step-2) cooperative cluster matching, and (Step-3) plan switching. The online phase triggers a hardware-based perception timeout pattern monitoring, whenever one of the safety critical perception software modules has a timeout (e.g. LiDAR); the monitoring hardware counts the

number of continuous timeouts across the subsequent LiDAR sampling intervals to identify whether timeout happens accidentally or continuously, e.g., over a user-defined h continuous timeouts. Based on our field study, LiDAR perception timeout happens either accidentally for a few times or continuously for more than 100 frames. Therefore, we set h to be 100. If continuous timeout happens, the online phase performs the next two steps based on the following equations:

$$index = \underset{i}{\operatorname{argmin}} \|F_{current} - F_i\| \tag{3.10}$$

$$p_i = D[index] \tag{3.11}$$

where $F_{current} = S_{current} \oplus K_{current}$. This is the current feature vector, which is the concatenation of current obstacle density distribution vector $S_{current}$ and current timeout pattern $K_{current}$; $\oplus$ is the concatenation operator. F_i is the feature vector of the i_{th} cluster, which consists of the obstacle density distribution vector and timeout pattern of the i_{th} cluster. Assuming road condition and traffic pattern do not dramatically change within a certain amount of time, the online resource management software will scan the features of all clusters and identify a matching cluster, by finding the cluster with the closest feature as the current one.

Hardware and software implementation. To meet the real-time requirement of online resource management, we implement Step-1 in CPU hardware by adding a set of comparators and counters as shown in Figure 3.10. The comparators track the timeouts; the counters accumulate the number of continuous timeouts. In our AV system, an 8-bit counter is sufficient to capture the continuous timeout patterns. Step-2 and Step-3 are triggered once Step-1 detects continuous timeouts. As a result, these Step-2 and Step-3 are performed infrequently. As such, we implement the primary functionality of these two steps by software as a load balancer [86, 100] in CPU; it acts as a reverse proxy of software module scheduling across CPU and GPU. The user-level implementation imposes negligible performance overhead demonstrated by our experiment results, due to the infrequent cluster matching and resource management plan switching. In our

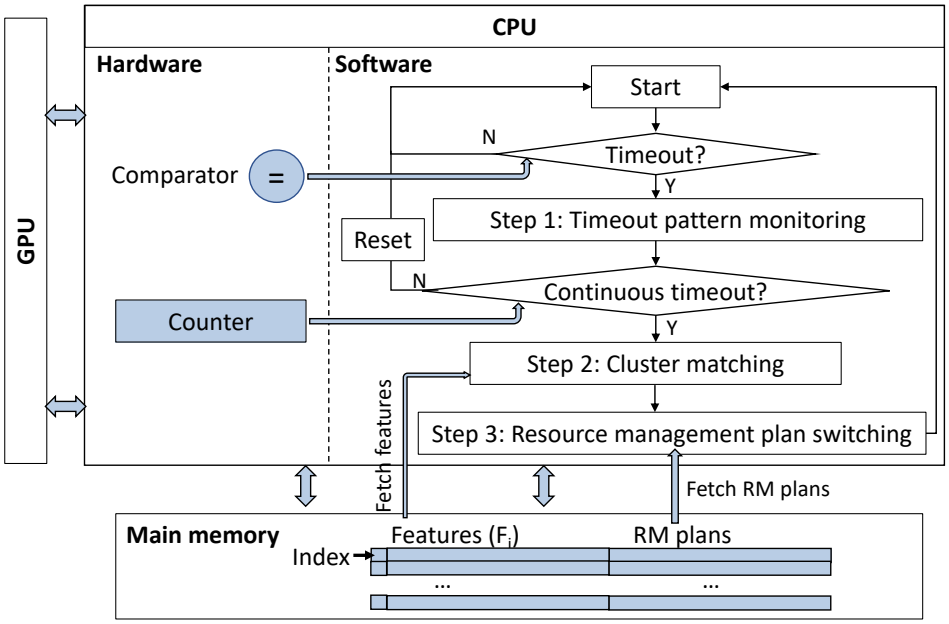


Figure 3.10. AV computing system design that performs safety-aware online monitoring and scheduling.

AV system, the software modules that can execute on both CPU and GPU are implemented and compiled with both versions. We store the indices generated during Step-2 in CPU main memory. Step-3 reads the memory to identify the resource management plan with the current feature.

3.8 Evaluation

Our evaluation employs two simulators. First, to evaluate safety score, we use an in-house AV simulator, which replays and analyzes road testing data[4]. Second, we use an in-house computer architecture simulator to evaluate the effectiveness of the online resource monitoring and scheduling in our resource management scheme. We compare three computing system architecture configurations and resource management policies:

- **CPU** – Executing all software modules on CPU;
- **CPU+GPU** – The native heterogeneous AV system design. It employs tail latency to determine the priority of software module execution in resource management, with the GPU to accelerate

[4]The simulator is also used by the company to analyze routine road testing data.

all deep learning software modules.

- **Resource management** – Heterogeneous AV computing system design with our resource management scheme. The results take into account the performance overhead of the online monitoring and scheduling phase.

3.8.1 The Need for Safety Score

Our safety-score, which is rigorously derived from the RSS model, measures the level of safety of AV system design. Furthermore, our AV simulator results show that the level of safety identified by our safety score is in line with the level of safety based on the probability of collisions.

To demonstrate that other traditional latency metrics do not accurately indicate the level of safety, we compare computing system resource management results guided by safety score, 95th percentile tail latency (99th percentile tail latency yields similar results), average latency, and maximum latency. We perform an exhaustive search of resource management plans for software modules. Due to space limitation, Figure 3.11 shows an example of five resource management plans. But the following observations hold with our exhaustive search. For fair comparison, we present the results obtained under the same set of obstacle distributions.

The results (the higher the better) show that safety-score-guided resource management leads to entirely different results than other metrics. For example, the safety score indicates that we should choose *RM-1*, whereas *RM-2*, *RM-3*, and *RM-4* leads to the best tail latency, average latency, and maximum latency, respectively. Therefore, guiding the computing system design by these traditional latency metrics can be misleading; it is critical to adopt safety score.

3.8.2 Latency Model Analysis

Pearson correlation coefficient. We adopt the Pearson correlation coefficient [40] to evaluate the correlation between LiDAR perception latency and obstacle density distribution. Figure 3.12 depicts a heat map of Pearson correlation coefficients distributed in an obstacle

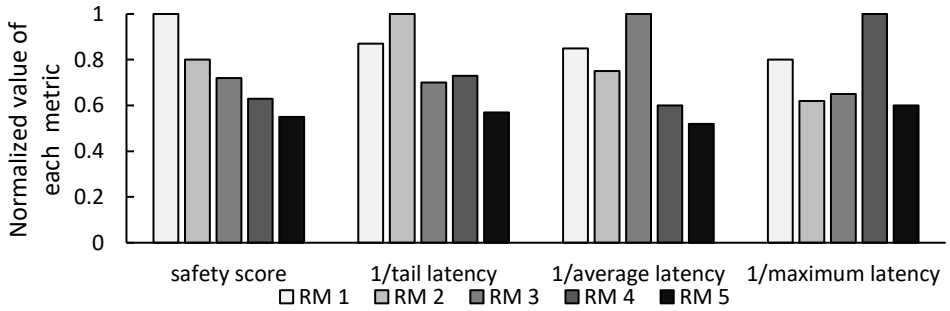


Figure 3.11. Resource management (RM) guided by various metrics (y-axis is normalized to the maximum of each metric).

count map hierarchy, based on real driving scenarios during our field study. Instead of the count of obstacles, each grid in Figure 3.12 stores a Pearson correlation coefficient. The coefficient measures the correlation between obstacle density and LiDAR perception latency: the higher the coefficient, the larger the correlation between the two. Figure 3.12 uses colors to express Pearson correlation coefficient values: the darker the color, the larger the value. As such, in the areas with darker colors, LiDAR perception latency is more susceptible to obstacle density. We make three observations from Figure 3.12. First, LiDAR perception latency is more susceptible to nearby obstacles than those far away. This is in line with our field study observation. Second, both the top left and top right areas have high Pearson coefficient. This may be caused by heavy horizontal traffic through an intersection during rush hours. Finally, LiDAR perception latency is more sensitive to the density than the distribution of obstacles.

Model accuracy. We use mean squared error (MSE) [138] (the lower the better) to evaluate the accuracy of our perception latency model quantitatively. We investigate our latency model under a variety of driving scenarios, such as morning and afternoon, rush hours and non-rush hours, and local roads and express ways. The proposed latency model consistently provides high accuracy with an average MSE as low as 1.7×10^{-4}.

Model coefficient. We borrow the above heat map methodology to analyze the coefficients in our perception latency model. Figure 3.13 shows the coefficients obtained based on our field study data. We make two observations. First, most coefficients are non-zero. Second,

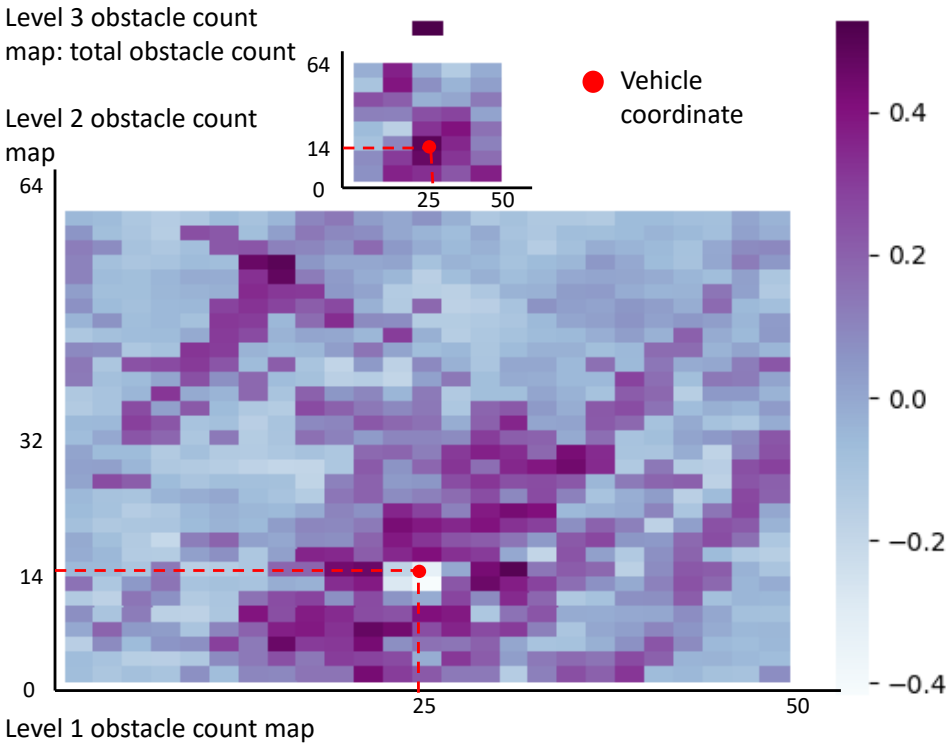


Figure 3.12. Heat map of Pearson correlation coefficient for obstacle count map hierarchy.

coefficients of different terms in Equation 3.8 have a significant variation on values. The coefficients of lower-order terms have larger values than those of higher-order terms. For instance, $\vec{a}$ (Figure 3.13(a)), which are the coefficients of the highest order term x^2 in Equation 3.8, has the smallest values at a scale of 10^{-8}.

3.8.3 Resource Management Analysis

Figure 3.14 compares the safety score of three different AV system architecture configurations and resource management policies. Compared with *CPU* and *CPU+GPU*, *Resource Management* achieves the highest safety score, which is 1.3× and 4.6× higher than *CPU+GPU* and *CPU*, respectively.

Beyond improving safety, *Resource Management* also improves computing system performance and energy efficiency. Figure 3.15 compares the energy-delay product (EDP) of various configurations. *Resource Management* leads to the lowest EDP in both LiDAR

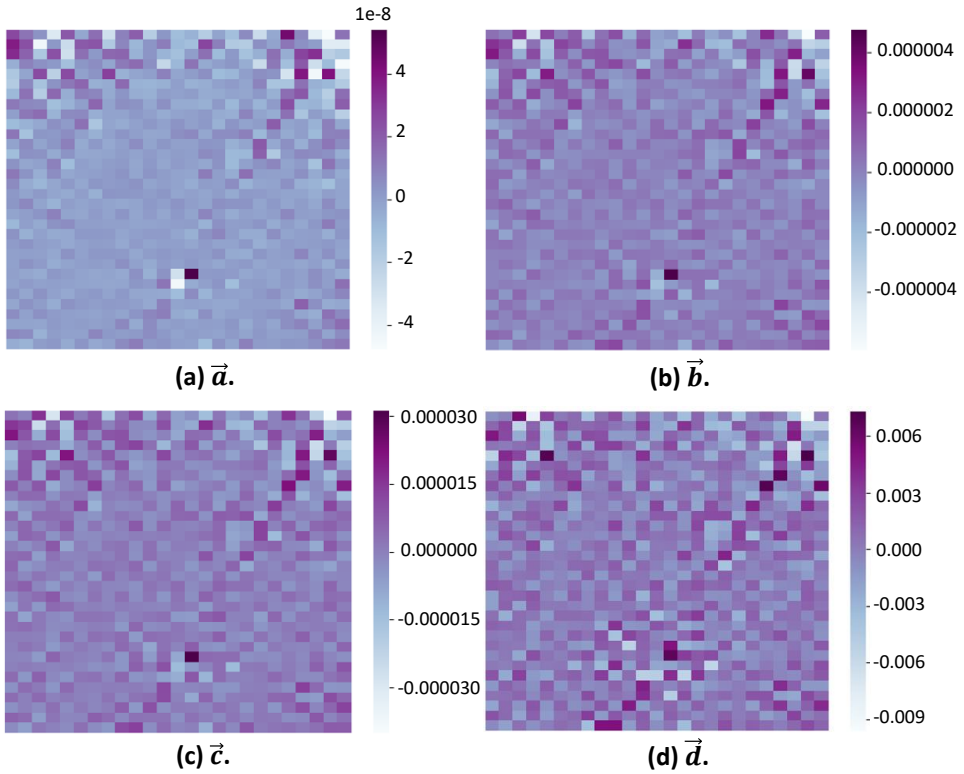


Figure 3.13. Heat maps of perception latency model coefficients $\vec{a}$, $\vec{b}$, $\vec{c}$, and $\vec{d}$.

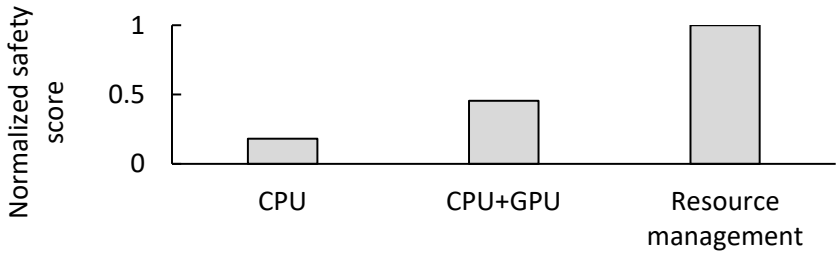


Figure 3.14. Safety score with various architecture configurations.

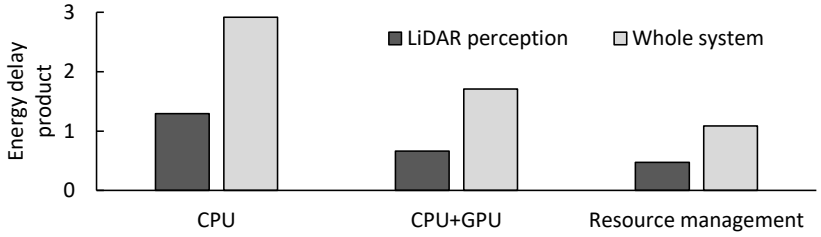


Figure 3.15. Energy efficiency of various AV computing system designs.

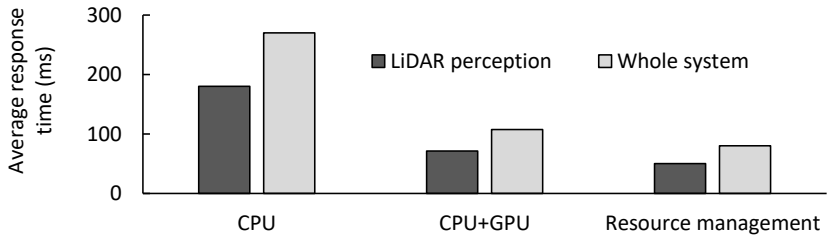


Figure 3.16. Average response time of various computing system designs.

perception alone and the whole computing system. Our hardware modifications to implement online resource management only impose less than 1% of total system energy consumption. Figure 3.16 compares the average response time of various configurations. Compared with *CPU* and GPU+CPU, *Resource management* reduces the response time averaged throughout our field study data by 2.4× and 35%, respectively; LiDAR perception latency is reduced by 2.6× and 45% on average.

3.9 Related Work

In this section, we discuss related works on AV nominal safety policies, safety-aware AV system design, and heterogeneous computing system design.

AV nominal safety policies. Two recent studies, RSS [129] and SFF [106], investigated AV nominal safety policies. Both studies define the high-level concept of AV nominal safety. Our study is in line with this high-level safety concept. Unfortunately, neither of the studies investigates safety requirement on computing systems and architecture design. SFF [106] focuses on policies that ensure safe driving decisions, e.g., braking, acceleration, and steering, under given limitations, such as perception visibility and AV system response time. SFF guides the computing system software algorithm design to enforce safety policies. But the study does not provide policies on timely driving decision making, which is one of the essential goals of computing systems and architecture design. RSS [129] develops a set of general rules to ensure collision avoidance, by enforcing that the AV keeps a safe distance with surrounding vehicles.

To this end, the safety rules treat the AV and surrounding vehicles as objects with given velocity and acceleration, without implications on the requirement of AV internal systems (including the computing system). Our study focuses on the nominal safety requirements of AV computing systems and architecture design.

Safety-aware AV system design. Most previous safety-aware AV system designs focus on developing software module algorithms [29, 49, 59, 60, 66, 83, 84, 113]. For instance, recent works improve the latency and precision of LiDAR- and camera-based perception, by leveraging video sensing, laser rangefinders, light-stripe range-finder to monitor all around the autonomous driving vehicle to improve safety with a map-based fusion system [32, 83, 113]. Several studies enhance the quality of driving decisions by optimizing perception and planning software algorithms [59, 60]. Previous works also investigate ways of effectively responding to uncertainty at real-time [29, 66, 84] and path planning to avoid static obstacles [49].

AV computing system design. Most previous studies on AV systems and architecture design focus on accelerator design and energy efficiency improvement of deep learning or computer vision software execution [74, 92]. The performance and energy optimizations are guided by standard performance metrics. Safety requirement is not adequately considered in most of these studies. As our evaluation demonstrates, safety-aware computing system design leads to different architecture and systems designs compared with guided by traditional design metrics. However, our metric and latency model can also be used to guide deep learning and computer vision accelerator design in AV systems.

Heterogeneous computing system design. AV computing systems adopt sophisticated heterogeneous architectures used in real-time scenarios. Prior works on real-time heterogeneous computing systems [45, 51, 133] focus on embedded processors, rather than sophisticated architectures. Previous studies that investigate server-grade CPU and GPU processor based systems focus on distributed and data center applications [38, 39, 50, 78, 81, 96]; in such cases, scalability is a more critical issue than single-system resource limitation. Furthermore, neither

type of heterogeneous system use cases has the level of safety requirements as autonomous driving.

3.10 Conclusion

In summary, this chapter proposes the safety score and the perception latency model to guide safety-aware AV computing system design. Furthermore, we elaborated detailed AV computing system architecture design and workloads, as well as discussing a set of implications of safety-aware AV system design based on a field study with industrial AVs, and demonstrate the use of our safety score and latency model with a safety-aware heterogeneous computation resource management scheme. As high automation vehicles are still at the early stages of development, we believe our work only scratched the surface of safety-aware AV system design. Substantial follow-up work is required to enhance the computing system design towards safe autonomous driving.

Chapter 3 contains material from "Driving Scenario Perception-Aware Computing System Design in Autonomous Vehicles", which appears in IEEE International Conference on Computer Design (ICCD), Oct, 2020, "Safety Score: A Quantitative Approach to Guiding Safety-Aware Autonomous Vehicle Computing System Design", which appears in IEEE Intelligent Vehicles Symposium, Oct, 2020 and "Towards Safety-Aware Computing System Design in Autonomous Vehicles", which appears in arXiv repo. All these three papers are worked by Hengyu Zhao, Yubo Zhang, Pingfan Meng, Hui Shi, Li Erran Li, Tiancheng Lou, and Jishen Zhao. The dissertation author was the primary investigator and author of these materials.

Chapter 4

Suraksha: A Framework to Validate AV Safety and Study the Safety Implications of Perception Design Choices

4.1 Introduction

Vehicles that perform some or all driving tasks autonomously are appearing on roads led by the continuing evolution of automotive and computer technologies [76, 80, 88]. An autonomous vehicle (AV) employs a set of sensors to sense surroundings, a set of software algorithms to process the sensor data and make driving decisions, and a hardware platform that executes the software algorithms in real-time. The AV system performs tasks that are similar to that of a human driver of a conventional vehicle such as perception, localization, planning, and control [93].

AV products must meet stringent safety requirements [69, 102, 118, 150]. AV safety issues stem from numerous sources, such as system design defects, software bugs, or reckless road users [37]. To avoid safety hazards, industrial AV technology developers recently proposed several design standards [70, 71] and safety models [106, 129]. As part of the AV design process, the hazard analysis and risk assessment (HARA) is performed to drive the end-to-end safety requirements, which are then converted to component-level requirements [69, 101]. While component-level testing is performed, no clear process exists to validate these requirements and

analyze the effect they have on AV safety. Without a methodology or framework to analyze the sensitivities of these component-level requirements on AV safety, the designed system may be over-provisioned for some components but compromise the overall functionality (by not providing sufficient resources for other tasks), and may need more area and power.

Perception is one of the most compute-intensive AV tasks in an AV stack [36, 42, 87, 139, 149]. Perception must process the data from the sensors and detect objects both at sufficiently high precision and low latency [92, 148]. Little research has been conducted to understand the perception requirements, trade-offs offered by different design choices, and implications they may have on safe and optimal system design. For example, for a given AV hardware platform, should the perception task provision lower camera frame per second (FPS) with a high accuracy obstacle perception model or higher camera FPS with a faster but slightly less accurate perception module? Moreover, perception requirements may vary significantly with driving scenarios. For example, the perception requirements for safe driving at 25 MPH on a straight road with no obstacles should be less than a scenario in which the AV is traveling at 45 MPH and reacting to a suddenly-appearing obstacle.

During the AV development process, a set of driving scenarios are used to test the system. These scenarios are based on an operational design domain (ODD [79, 123]) to limit the scope of testing. While the processes described in standards (e.g., ISO 26262) suggest the use of ODDs, no guidance to select driving conditions for the driving scenarios are provided (e.g., what speeds should one use for actors in the scenarios). AV stacks can be configured with different component-level parameters and evaluated in a set of driving scenarios. To validate AV safety, metrics that are independent of the scenarios and AV's internal states are required. However, such safety quantification metrics have not been established.

To address these challenges, we propose Suraksha (Suraksha means safety in Sanskrit), an AV safety evaluation framework that automates the analysis of the safety effects of different component-level design choices. Suraksha automatically generates AV versions based on user-specified low-level component parameters, and driving scenarios based on the desired difficulty

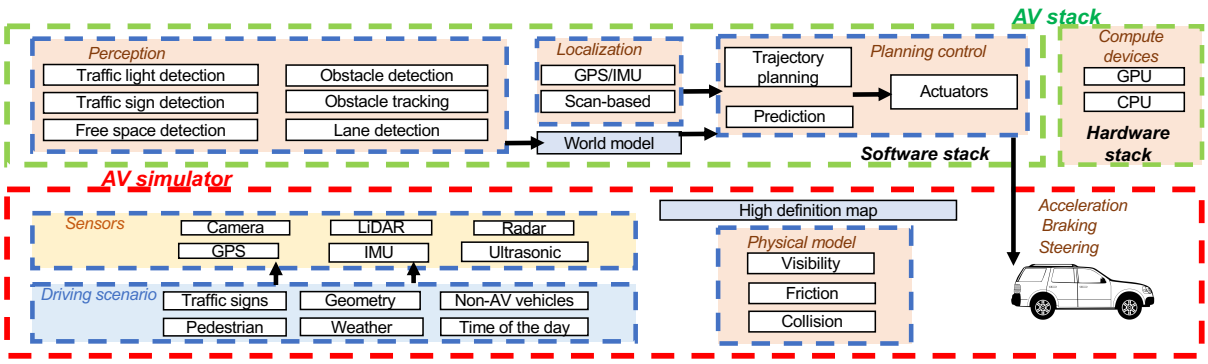


Figure 4.1. AV stack and AV simulator.

levels. It quantifies safety using a set of scenario-independent metrics, and analyzes safety sensitivity of different component parameters. Automation of all these steps accelerates the discovery of the performance vs. safety trade-offs offered by different design choices. The insights derived from the trade-offs and sensitivity analysis of component-level design choices enabled by Suraksha allow the AV engineers to better design the system for performance and safety.

With the focus on the perception system, we employ Suraksha to evaluate the safety effects by degrading perception quality with multiple component-level parameters, including camera FPS and neural network model precision. To further understand the perception quality limits that allow for safe driving in the selected scenarios, we developed models that introduce inaccuracies directly into the perceived surroundings before it is sent to the planning and control tasks. We study the safety tolerance in an industry-implemented AV by injecting noise in every frame's perception output and introducing intermittent delay in perception. The results interestingly reveal that the tested AV tolerated significant world model inaccuracies even in hard driving scenarios where hard braking was originally required to remain safe. These findings indicate potential system optimization opportunities and highlights the benefits of using Suraksha. In summary, the chapter makes the following key contributions.

- We present an automated AV safety evaluation framework called Suraksha that quantitatively analyzes safety sensitivity of different AV versions configured using component-level

parameters while running driving scenarios based on a user-selected difficulty level.

- We employ Suraksha to study the safety effects of deteriorating perception quality by changing component-level parameters. We also introduce inaccuracies to the perceived world model directly to study the limits of the AV's perception requirements for safe driving.
- The results demonstrate that an industry-developed AV tolerates small per-frame inaccuracies and reduced camera FPS even in the hardest tested scenarios. We show that the Suraksha's sensitivity analysis identifies the perception parameters that affect safety the most when altered.

4.2 Background and Challenges

4.2.1 AV Stack

The top part of Figure 4.1 depicts a typical AV stack, which consists of software and hardware stacks. The software stack performs three major AV tasks – perception, localization, and planning control. Perception detects and interprets surrounding static (e.g., lane lines, trees, and traffic lights) and moving (e.g. other vehicles and pedestrians) objects or obstacles, based on camera, radar, ultrasonic sensors, and/or light detecting and ranging (LiDAR) [93]. Localization identifies the current AV location in a high-precision map using GPS/IMU sensors and/or scan-based methods [144]. Planning control employs perception and localization results to plan for the trajectory, predict other obstacles' behaviors, and generate actuator commands to the vehicle controller. The hardware stack typically adopts multiple sensors and a heterogeneous computing system with CPUs, GPUs, and accelerators to execute the software modules [93].

4.2.2 AV Safety Validation

Safety engineering and testing: An AV system is often tested and validated in constrained driving situations defined by an operational design domain (ODD) [79, 123]. An ODD may include but not limit to environmental, geographical, time-of-day, and traffic constraints.

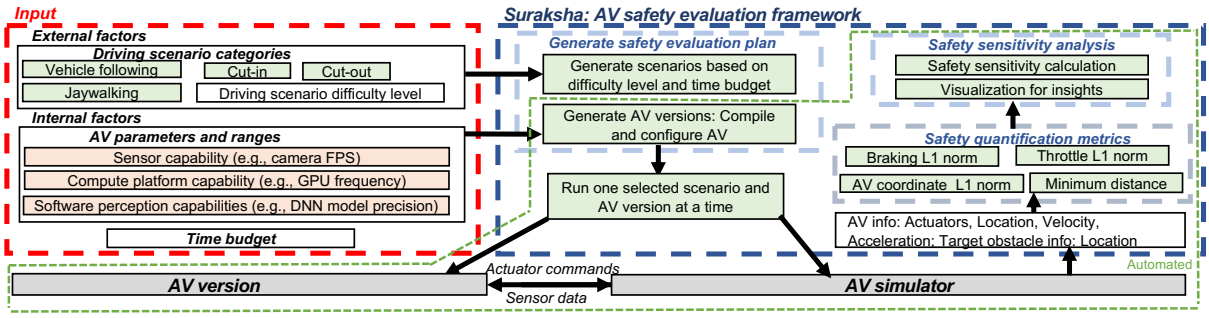


Figure 4.2. Overview of our AV safety evaluation framework Suraksha.

AV designers specify a suite of scenario categories (e.g., cut-in and cut-out) based on the selected ODD. These scenarios aim to cover different driving situations to test whether the AV under test follows traffic rules and etiquette while keeping the AV safe.

An AV is also tested under hazardous situations. ISO 26262 safety standard requires a Hazard Analysis and Risk Assessment (HARA) to be performed to determine vehicle level hazards. This process guides the safety engineers towards safety goals that are then used to create functional safety requirements [69, 101]. These requirements guide the system development process, which is then decomposed into hardware and software development processes. The standard defines a V-model for the safety requirements to be sufficiently fulfilled by the software architectural design and testing. While emphasis is given on unit-level and inter-unit testing, no guidance or process is provided to validate or analyze the effects of component-level requirements on AV safety under challenging operating conditions (not covered by HARA and ODD analysis). Moreover, these standards do not provide any guidance on improving resource usage while conducting AV tasks.

AV simulation: AV simulators are widely used by industry and academia [54, 112, 121] to develop techniques to test and validate AV systems. Simulation-based testing is salable and safe. The bottom part of Figure 4.1 illustrates a typical AV simulator. An AV simulator has four key tasks. (1) It models the driving scenarios. A *driving scenario* consists of the information about the surroundings, such as traffic lights and signs, pedestrians, traffic, weather, and time of the day (e.g., sun position). (2) It models sensors such as cameras, LiDAR, radar, GPS/IMU, and

Ultrasonic devices, and offers an interface for the AV to receive the sensor data. (3) It models the physical characteristics of the real world, such as visibility due to a cloudy weather, less friction due to wet roads, and collision force. (4) It models vehicle dynamics and offers an interface to receive actuator commands (e.g., steering) from an AV.

Real-world testing: Real-world testing on public roads will be employed by AV developers to ensure that their designs are fully operational and safe. While real-world testing allows for practical and comprehensive safety investigations [76, 88], it is expensive to perform at the design stage and difficult to scale for analyzing and testing a large variety of design choices.

4.2.3 Challenges

Despite the current development in AV system design and safety validation, substantial challenges still remain in designing a resource-efficient and safe AV. First, as AVs perform the complex tasks and are composed of many low-level components, assigning requirements to such components and ensuring an efficient and optimal assignment for overall AV safety is challenging. Second, component-level metrics such as DNN model precision, which has been used by prior work [89, 140], do not provide sufficient insights into the safety impact of different component-level design choices. Third, in a specific ODD, while AV safety depends on driving scenarios [104], there is no existing guidance on how to select a limited set of realistic but challenging scenarios that can be used to study AV's performance. Finally, despite perception being a system resource demanding task and having many low-level design choices, little research has been conducted to understand AV safety effects due to low-level decisions (e.g., using a reduced precision model with high frame rate camera). We address these challenges by developing Suraksha and employing it on an industry AV system to improve our understanding of the perception component-level requirements and allow AV designers to make better trade-offs.

4.3 Suraksha Framework

To address the challenges listed in Section 4.2.3, we develop Suraksha, an AV safety evaluation framework that quantifies safety for different AV versions. An **AV version** is an AV system compiled with specific options or configured with specific parameter settings, e.g., FP16 models running on an 800MHz GPU processing 30 camera FPS. Suraksha interacts with an AV simulator and an AV stack based on the system designers' inputs; it provides feedback in the form of safety analysis. Figure 4.2 summarizes the components of Suraksha. Suraksha consists of three main components – methods to generate AV versions and driving scenarios for safety evaluation, safety quantification metrics for a single experiment (for a chosen AV version and driving scenario combination), and safety sensitivity analysis across a set of scenarios for the different AV versions being considered. The inputs to Suraksha are the following: (1) a set of AV parameters that define different AV versions, (2) a set of driving scenario categories based on user-selected ODD, and (3) scenario difficulty level, which is used by Suraksha to initialize conditions, e.g., speeds of actors in the scenarios. The outputs of Suraksha are safety quantification metrics for different AV versions and safety sensitivity quantification for each of the AV parameters; these results will aid the AV designers to identify sensitive factors and optimize the system for safety and performance.

4.3.1 Generating AV Versions and Driving Scenarios

Suraksha receives high-level inputs such as AV design parameters and a set of driving scenario categories the designers want to use for AV safety analysis, and creates an evaluation plan. While creating a plan, Suraksha considers (1) the number of AV versions that can be generated based on the selected parameters and their setting ranges (changing one parameter at a time), (2) the number of driving scenario categories provided for which challenging scenario settings need to be generated, (3) the duration of each of the scenario categories, and (4) the provided time-budget. Each experiment in the created plan specifies an AV version and a driving

scenario for simulation. While we primarily investigate changing one parameter setting at a time, which helped us uncover new insights, the plan generation algorithm can also consider AV versions that change multiple parameter settings. Such an investigation increases the exploration space significantly. We study a preliminary strategy to explore this space in Section 4.6.6, paving the way for future research in this direction.

AV version generation: For a given experiment, Suraksha generates an AV version using a set of selected AV parameters and settings. It compiles the AV software, configures it, or changes hardware configuration, as needed.

Driving scenario generation: An input to Suraksha is a high-level description of a scenario category, without the details such as the speeds of actors and their relative locations. For a given scenario category, the goal is to generate a scenario that is challenging but does not include an accident that is unavoidable. Based on the user-specified difficulty level, Suraksha generates driving scenarios by setting initial conditions, i.e., velocities of different actors and their relative positions on the road. To achieve the best evaluation efficiency, users may select categories based on an ODD and the following principles. (1) The selected scenario category should be realistic enough to be encountered by human drivers on the road. (2) While being realistic, the scenarios should be more challenging than most driving scenarios to expose safety issues.

4.3.2 Safety Quantification Metrics

As the goal is to study AV safety effects by changing AV system parameters and settings, the low-level metrics such as system latency, sensor processing rate, and system power consumption are inappropriate as they fail to capture high-level safety effects. The high-level AV behavior may change less compared to internal factors. So, we consider the following high-level behavioral aspects (also summarized in Table 4.1).

Distance from obstacles: An AV's goal is to avoid collisions and stay safe on the roads. There can be many actors around the AV in a driving scenario, and the distance from each of

Table 4.1. Safety quantification metrics.

Metric	Description
Minimum distance	The minimum distance between the AV and any target during the simulation
AV coordinate (x, y, z) L1 norm	L1 norm of AV coordinate (x, y, z) between default configuration and any AV version
Braking value L1 norm	L1 norm [134] of braking value between default configuration and any AV version
Throttle value L1 norm	L1 norm of throttle value between default configuration and any AV version

them can be tracked over time. The minimum distance to the closest actor in the entire scenario is an indicator to how close the AV came to a collision. If the minimum distance is below a threshold (e.g., 0), the AV collides with an obstacle. We use minimum distance to the closet actor during a scenario (or simply minimum distance) as a key safety metric.

AV trajectory: When an AV system changes, the AV's driving behavior and trajectory can also change. This change is quantified by computing an L1 norm using the AV's coordinates over time in a driving scenario and the coordinate values from a different execution using the same scenario but with the default (or baseline) AV version.

Actuator values: Actuators include brake, throttle, and steering. If the AV stack changes, these values will also change, and can eventually affect safety. We quantify the change in the actuator values due to a change in the AV version by computing L1 norm per actuator.

4.3.3 Safety Sensitivity Analysis

For each of the AV parameters used to generate different AV versions, Suraksha also computes a sensitivity metric by leveraging the previously computed safety metrics (e.g., minimum distance) for different driving scenarios. As part of the sensitivity metric computation, Suraksha computes the relative safety metric value difference between two neighboring settings of the selected AV parameter; we call it *delta*. Delta reflects the relative change in safety due to a change in a setting to the next or previous AV parameter setting. The sensitivity metric is obtained by aggregating the delta values. In this aggregation, worst-case and best-case delta values can be

weighted differently and can lead to different insights. So we compute *average* delta that weighs all the outcomes equally, *minimum* delta that considers the worst-case outcome only, and the 10^{th} *percentile* that weighs the worst-case outcomes more than the best-case outcome, assuming a lower metric value is worse (e.g., lower minimum distance is less safe and hence is considered worse).

4.4 Suraksha Implementation

4.4.1 Generating AV Versions and Driving Scenarios

AV version: Suraksha investigates AV safety by changing one AV parameter at a time. For one experiment, Suraksha sets the AV parameter by modifying one or multiple of the following: AV configuration file, sensor rig file, AV software compilation flags, and hardware settings (e.g., GPU clock setting). Additionally, we experiment by changing two AV parameters at a time (e.g., FPS and model precision), where a sensitive parameter is set based on the value below which AV becomes unsafe for the considered scenarios.

Driving scenarios: According to the principles described in Section 4.3.1, we select an urban expressway as the ODD and select four scenario categories that are realistic, encountered occasionally on-road, and can be set to create a challenging condition for the AV, i.e., the AV has to apply hard brakes after detecting a suddenly appearing obstacle. The four scenario categories, which are inspired by NCAP scenarios [104], are as follows. In the first scenario (*Vehicle following*), a vehicle traveling ahead of the AV brakes suddenly. In the second scenario (*Cut-in*), a vehicle traveling at a slower speed changes lane ahead of the AV from an adjacent lane to the lane in which the AV is driving. In the third scenario (*Cut-out*), a vehicle ahead of the AV moves out to suddenly reveal a static obstacle. In the fourth scenario (*Jaywalking*), a jaywalker crossing the road appears from behind a car stopped in an adjacent lane ahead of the AV. In all these scenarios an obstacle is revealed suddenly, and the AV needs to perceive the obstacle and apply hard braking to avoid a collision. These scenarios are summarized in Figure 4.3.

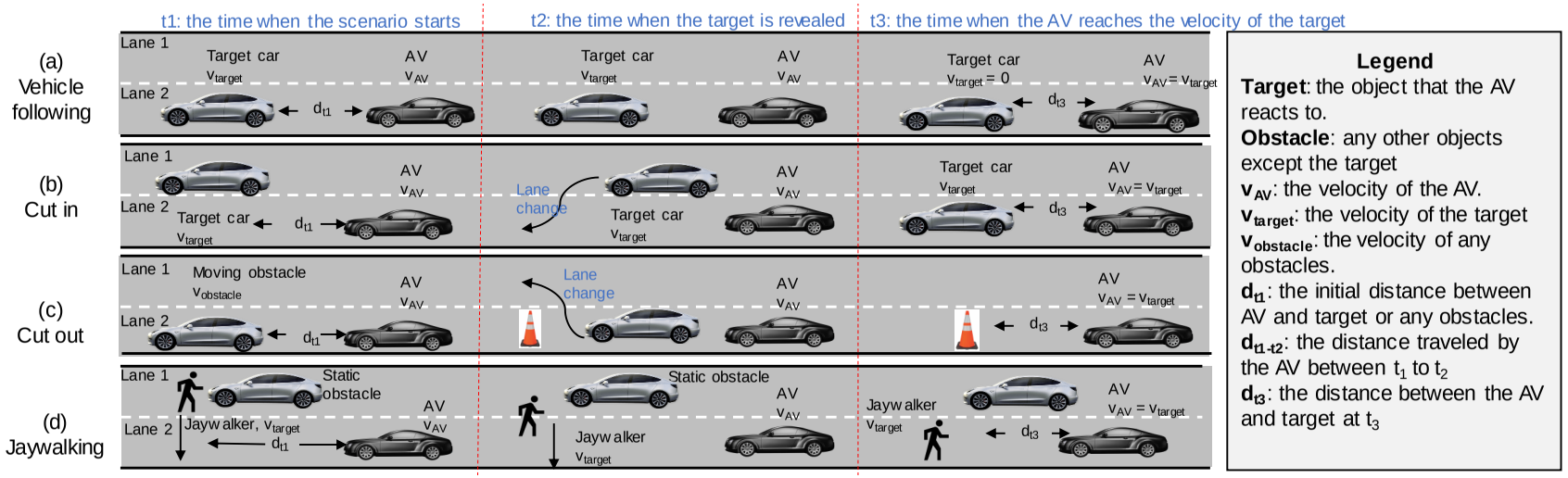


Figure 4.3. Four driving scenario categories for an urban expressway ODD.

For each driving scenario, we initialize locations and velocities for each of the actors (e.g., $v_{AV,t1}$ and $v_{target,t1}$ will be the velocities for the AV and target obstacle at the start of the scenario, $t1$). We define t_2 as the time when the AV starts reacting to the target (e.g., traffic cone or jaywalker in scenarios (c) and (d) in Figure 4.3 and t_3 as the time when the AV reaches the target's speed. We define braking distance, d_{t2-t3}, as the distance travelled by the AV between t_2 and t_3. Scenarios with shorter d_{t2-t3} are more challenging for AVs, because the AV needs to apply harder braking. Higher braking results in higher deceleration, which may result in an uncomfortable drive.

The AV can apply uniform or non-uniform braking, so the braking distance can vary between AV versions based on the differences in perception and driving policies (some AV versions may stop closer to the obstacle than the others). We employ Equation 4.1 to calculate the average braking acceleration (a_{avg}) for a given scenario, where $v_{AV,t2}$ is the velocity of the AV at t_2 and $v_{target,t3}$ is the velocity of the target at t_3.

$$a_{avg} = \frac{v_{AV,t2}^2 - v_{target,t3}^2}{2d_{t2-t3}} \tag{4.1}$$

We simulate the baseline AV on a scenario (with an initial assignment) and categorize it by comparing a_{avg} with the gravitational acceleration ($g = 9.8m/s^2$). We label the scenario hard if $a_{avg} > \frac{g}{2}$, moderate if $\frac{g}{4} < a_{avg} \leq \frac{g}{2}$, and easy if $a_{avg} \leq \frac{g}{4}$. Based on the user-specified objective, Suraksha can select a desired mix of scenarios with different difficulty levels.

4.4.2 Safety Quantification Metrics

For a given scenario, we collect the coordinates of all the actors (including the AV) along with the AV's actuator values from the simulator at a fixed frequency, which is 60Hz in our setup. We use this information to compute our safety metrics, which are the minimum distance between the AV and any obstacle in the scenario at any timestep, and the L1 norms computed using AV coordinates, braking values, and throttle values between an AV version being evaluated

and the baseline system (as described in Section 4.3.2). For the baseline system, we use the out-of-the-box AV version with the default values assigned to the AV stack parameters. We show the default values in Table 4.2.

We calculate minimum distance in meters. We store the AV coordinate and actuator data from the scenario executed with the baseline AV for the L1 norm computation with different AV versions. We calculate the difference between values obtained from the two simulations (using the baseline AV and a new AV version) at each timestep in the scenario. We accumulate the differences over all the timesteps and normalize it by dividing it by the number of timesteps (N). This calculation is described in Equation 4.2, where $q_{v_e,t}$ and $q_{v_b,t}$ represent measured actuator or coordinate values at timestep t for the two AV versions (v_e refers to the AV version that will be evaluated and v_b refers to the baseline AV version), respectively. If a simulation ends sooner at timestep N_s, with $N_s < N$ (due to a collision, for example), we compute the L1 norm only for the N_s timesteps.

$$L1\ norm = \frac{\sum^{N} |q_{v_e,t} - q_{v_b,t}|}{N} \tag{4.2}$$

4.4.3 Safety Sensitivity Implementation

Suraksha obtains the AV parameter list and their setting ranges from the user. For each parameter and setting combination, Suraksha conducts one experiment for each of the selected driving scenarios. We compute delta between the neighboring settings to analyze sensitivity of a parameter using Equation 4.3. In this equation, m refers to a metric value and it can be any of four metrics listed in Table 4.1 (e.g., minimum distance); d is the driving scenario from which the reading is obtained; p is the design parameter (e.g., FPS); s_1 and s_2 are two neighbor settings for the parameter p; and s_1 is closer to the default setting (e.g., $s_1 = 15$ and $s_2 = 10$ with default

FPS = 30).

$$delta_{p,d} = \frac{m_{p,s_2,d} - m_{p,s_1,d}}{m_{p,s_1,d}} \tag{4.3}$$

To analyze sensitivity of a parameter p, we compute average, minimum, and 10^{th} percentile using the computed $delta_{p,d}$ across all the driving scenarios. We refer to them as $avg(delta_p)$, $min(delta_p)$, and $10^{th}percentile(delta_p)$. The higher the value, the more sensitive the parameter is. We also compute average, minimum, or 10^{th} percentile for parameter p using $m_{p,s_1,d}$ across all the settings of p and driving scenarios. We refer to them as $avg(m_p)$, $min(m_p)$, and $10^{th}percentile(m_p)$. For metrics where higher value is worse (e.g., L1 norm) we use avg, max, and $90^{th}percentile$ functions instead. As the final step, we visualize the computed values for the desired metric (e.g., $10^{th}percentile(delta_p)$ and $10^{th}percentile(m_p)$ for minimum distance) to gain insights into the sensitivity of p.

4.5 Perception Quality Requirements Analysis

The general Suraksha framework can be used to analyze the safety effects of using a different sensor suite, perception stack, localization algorithm, and path planning strategy. Once the AV designers parameterize the system for the components they want to study, Suraksha can configure the AV system for an automated experimentation and analysis. We focus on evaluating the effects of changing perception quality on AV safety. It has been demonstrated that the computational capability offered by hardware platform for complex perception stacks can be a bottleneck [87, 92, 139, 148], making it important to understand the effect of different perception parameters on AV safety to aid the development of an optimal and safer system design. We generate AV versions that degraded perception quality compared to the baseline AV version and evaluate their impact on safety. We adjust AV's perception quality by modifying AV's perception parameters and by directly modifying the perceived world model, which is the main part of the input to the planning and control logic. We summarize the AV parameters and inaccurate world

Table 4.2. Parameters and corruption models used to degrade perception.

Category	Parameter/model	Settings
Software parameter	CNN model precision	INT8, FP16 (default)
	CNN model version	v1 (default), v2
Hardware parameter	Camera FPS	1, 2, 3, 5, 6, 10, 15, 30 (default)
	GPU frequency (MHz)	300, 500, 800, 1000, 1200 (default)
Inaccurate world model prediction	Random noise in obstacle distance/velocity	[-0.1,0.1], {[-0.3,-0.1),(0.1,0.3]}, {[-0.5,-0.30),(0.3,0.5]}, {[-0.7,-0.5),(0.5,0.7]}, {[-0.9,-0.7),(0.7,0.9]}
	Positive noise in obstacle distance/velocity	[0, 0.1], (0.1, 0.3], (0.3, 0.5], (0.5, 0.7], (0.7, 0.9]
	Negative noise in obstacle distance/velocity	[-0.1,0], [-0.3,0], [-0.5,0], [-0.7,0], [-0.9,0]
	Obstacle perception delay (in frames) per 100 frames	10, 30, 50, 70, 90
	World model loss (in frames) per 100 frames	10, 30, 50, 70, 90

model perception models we use in this study below and in Table 4.2.

Software parameters: We select two parameters related to the object detection CNN model: model precision and model version. The model precision has two options – INT8 and FP16, each of which refers to the data type used by the optimized model during inference. The model accuracy of INT8 model is slightly lower than that of FP16. The model version has two options – v1 and v2, each of which are trained and optimized differently. Model v2 has higher precision than model v1.

Hardware parameters: We select two existing parameters to adjust the compute intensity – GPU frequency and camera FPS. We study six settings for the GPU frequency, which spans the valid GPU frequency range for the GPU used in the experiments (NVIDIA RTX 2080). We use eight settings for camera FPS. The maximum (and default) camera FPS is 30. Each of selected FPS settings is a divisor of 30.

Inaccurate world model prediction: While varying individual perception parameters (or a group of them) can inform the parameters critical to safety, it does not disclose the perception quality limits needed for safe operation, knowing which can lead to better perception system

design. To understand the perception needs better, we introduce inaccuracies directly in the world model. We study three types of inaccuracies – random noise, obstacle perception delay, and world model loss. The random noise models inaccurate obstacle perception models that may introduce higher noise; the perception delay model models stress on compute resource, for instance, hardware oversubscription or power throttling due to operating conditions can introduce intermittent delay; and the world model loss aims to model intermittent perception failures due to hardware or software issues. For a world object, we change the perceived location and velocity.

For the random noise model, we introduce noise in the target obstacle's location and velocity at *every timestep* during the scenario. We further consider three options – inject random noise including both positive and negative noise; inject positive noise such that the object is perceived farther and faster than its actual distance and speed, respectively; inject negative noise such that the object is perceived closer and slower than its actual distance and speed, respectively. For each of these options, we select five settings. A setting specifies the error range for the noise injection, which is injected at every timestep based on the selected range. Equation 4.4 shows how we inject the noise in the world model.

$$W_{inaccurate,t} = W_{perceived,t} \times (1 + e_t) \quad \begin{matrix} l_1 \leq e_t \leq l_2 \\ l_3 \leq e_t \leq l_4 \end{matrix} \tag{4.4}$$

$W_{perceived,t}$ and $W_{inaccurate,t}$ refer to the perceived world model at timestep t before and after the noise injection, respectively; e_t refers to the random noise value at time t selected based on the range setting; and l_1, l_2, l_3, l_4 define the range of the noise as listed in Table 4.2. For example, the second setting for random noise injection model in Table 4.2 sets $l_1 = -0.3, l_2 = -0.1, l_3 = 0.1, l_4 = 0.3$ and e_t is randomly generated from the ranges [-0.3,-0.1) and (0.1,0.3].

For the perception delay model, we consider five settings (shown in Table 4.2) that specify the number of frames (D) for which the perception output will be delayed every F frames

($D < F$). Equation 4.5 explains our delay model.

$$W_{inaccurate,f1} = W_{perceived,f2} \quad \begin{matrix} f2 = \left\lfloor \frac{f1}{F} \right\rfloor, f1 \bmod F = 1..D \\ f2 = f1, \quad f1 \bmod F = D..F \end{matrix} \tag{4.5}$$

To study world model loss, we consider five settings (shown in Table 4.2) that specify the number of frames (D) for which the target's information is lost (or removed) from the world model every F frames ($D < F$). Equation 4.6 explains this model.

$$\begin{aligned} W_{inaccurate,f} &= W_{perceived,f} \quad f \bmod F = D..F \\ W_{inaccurate,f} &= 0\ (Blank) \quad f \bmod F = 1..D \end{aligned} \tag{4.6}$$

As described earlier, the goal of these models is to understand the limits in perception inaccuracies introduced by the AV stack that are tolerated and do not result in AV safety issues. Models that capture the manifestations of random hardware errors (e.g., transient and permanent errors) or sensor errors (e.g., mud on camera or dead pixels in a camera) at the world model level can also be incorporated in Suraksha, to expand the scope of analysis, which is part of our future work.

4.6 Evaluation

4.6.1 Experimental Methodology

Simulation methodology and AV stack: We use NVIDIA DRIVE Sim to simulate the environment, driving scenarios, and AV response [112]. It provides a virtual sensor set (e.g., cameras, radars, LiDARs) to configure AV sensor rigs and an interface to forward the sensor data to the AV stack for perception. It also provides a simulation state monitoring interface to collect AV actuator values, and location and velocity of all the actors, which we use to calculate safety metrics defined in Section 4.3.2. We use NVIDIA's state-of-the-art Level-2 AV [110]. Level 2 autonomy provides partial automation of driving functions [124]. One of its key function is

Adaptive Cruise Control (ACC) [99], which is responsible for maintaining a safe distance while following a vehicle in the front. In our implementations, the AV stack employs one front-facing camera and one radar to perceive the obstacles. While the sensor configuration is scalable, we use this setup to analyze a common use-case.

Hardware: We execute the simulator and the AV stack on a PC with Intel(R) Core(TM) i7-5820K CPU with 32GB system memory and two discrete GPU cards. We run the simulator on an NVIDIA TITAN V (Volta) GPU [109] and the AV stack on an NVIDIA GeForce RTX 2080 (Turing) GPU [108]. Changing GPU frequency for RTX 2080 from 1200MHz to 300MHz, reduces computational throughput and increases the latency by 4× each. We do not change the memory frequency.

Safety quantification metrics: We use the minimum distance metric to determine a collision. The exact minimum distance for a collision depends on several factors – center of the vehicle, which is used to measure the distance between two vehicles; size of the car, and orientation of the car. While we account for the size of the AV and the other actor, we do not account for the orientation and hence use the following criteria to define a collision. We consider the AV has a collision when the minimum distance is less than 3m. The AV is safe when the minimum distance is more than the length of the AV (we use Ford Fusion as the AV, which is about 5m long), and it is close to obstacle when the minimum distance is between 3m and 5m. For AV trajectory change, we only show the L1 norm of the AV's y coordinate values because the AV drives along y-axis orientation in all four scenarios.

4.6.2 Driving Scenarios

As described in Sections 4.3.1 and 4.4.1, we select four categories of driving scenarios and generate scenarios for each category with different levels of difficulties. In Table 4.3, we show 12 driving scenarios, three scenarios per category to show the effect of the initial conditions on a_{avg}, which we use to assign the difficulty level. The higher the AV velocity when the target is revealed ($v_{AV,t2}$), the harder the scenario is, because the AV must perform harder braking to

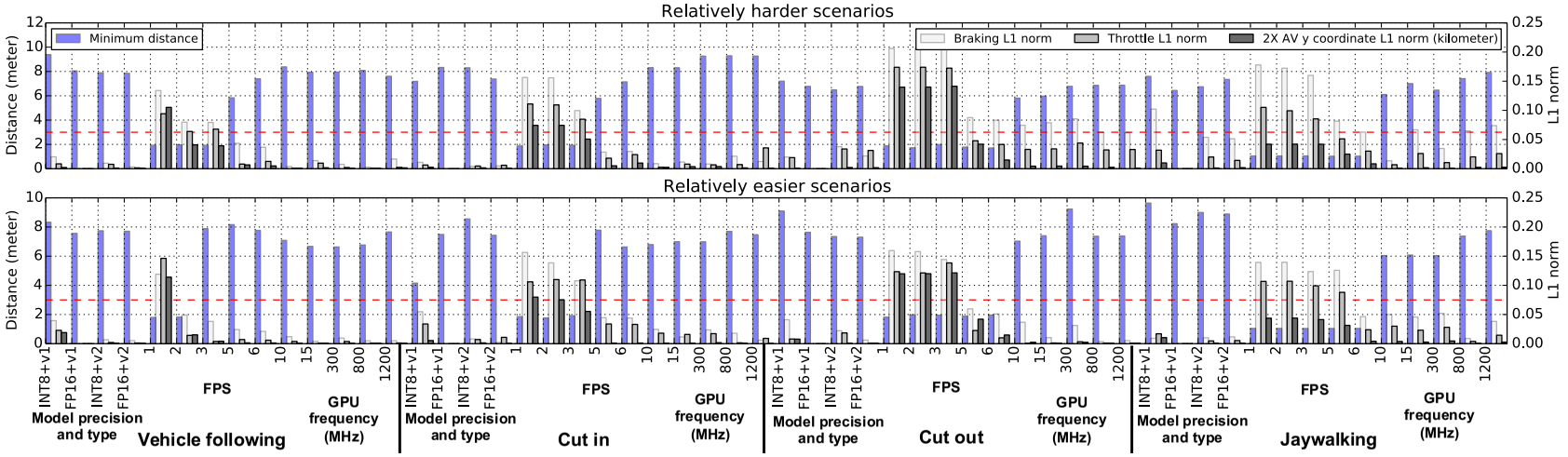


Figure 4.4. Effects on AV safety due to perception degradation with SW/HW parameters. For each scenario and parameter settings (x-axis), primary and secondary y-axes show the minimum distance and L1 norms, respectively. A blue bar (minimum distance) below the red dotted line indicates a collision.

Table 4.3. Driving scenario analysis. We select the scenarios in bold font for further analysis.

Category	#	d_{t1} (m)	$v_{AV,t2}$ (m/s)	$v_{target,t3}$ (m/s)	d_{t2-t3} (m)	a_{avg}	Difficulty
Vehicle following	**1**	**100**	**25.9**	**0**	**65.8**	**5.10**	**Hard**
	2	100	24.67	0	60.6	5.02	Hard
	3	**50**	**6.55**	**0**	**9.9**	**2.17**	**Easy**
Cut-in	**1**	**400**	**31.8**	**8.9**	**68.2**	**3.84**	**Moderate**
	2	330	25.9	8.9	51.3	2.82	Moderate
	3	**265**	**22**	**8.9**	**42.9**	**2**	**Easy**
Cut-out	**1**	**30**	**27.7**	**0**	**63.1**	**6.08**	**Hard**
	2	40	24.8	0	53	5.80	Hard
	3	**90**	**13.3**	**0**	**22.3**	**3.97**	**Moderate**
Jaywalking	**1**	**180**	**16.5**	**0**	**28.7**	**4.74**	**Moderate**
	2	230	15.2	0	26.4	4.38	Moderate
	3	**280**	**13.4**	**0**	**43.8**	**2.05**	**Easy**

avoid a collision. A harder scenario implies quick and accurate perception is required to brake in time and avoid the collision. We also find that the shorter the initial distance between the AV and the target (d_{t1}), the harder the scenario, for similar reasons. For analysis, we select one relatively harder and one relatively easier scenarios for each category, which are shown in bold font in Table 4.3.

4.6.3 Degrading Perception with SW and HW Parameters

In this section, we study how limiting software and hardware capabilities affect safety. Figure 4.4 shows the results by adjusting four perception parameters – model type, model precision, camera FPS, and GPU frequency. Minimum distance is plotted on the left y-axis, and whenever the value goes below the red dotted horizontal line (3m), we assume the AV is in a collision. L1 norms of the brake and throttle values compared to the baseline AV version are plotted on the secondary (right) y-axis. The range for brake and throttle values is [0,1]. L1 norm for the AV's y-coordinate values, which quantifies the change in the AV's trajectory, is also plotted on the secondary y-axis. It is measured in meters but we scale the values (to kilometers) to plot it on the same secondary y-axis. The L1 norm will be small if small changes are observed in many timesteps or large differences are observed in a few timesteps.

Model precision: The use of INT8 models results in slightly higher L1 norms for braking, throttle,

and y coordinate. The INT8 model used for perception is slightly less accurate than FP16, which means there might be some noise in obstacle perception. If an obstacle is perceived closer in some timesteps, the planning module may take a conservative approach and brake harder to avoid a potential collision, increasing the minimum distance. We observe a slight increase in the minimum distance in most scenarios we tested with INT8. The minimum distance may decrease if the error introduced by the less-accurate model always predicts the obstacle farther than its actual location, which may be the case with the easier cut-in scenario. While we observe some differences, no safety concerns are observed.

Model version: Results show that safety with v2 is similar to v1 at FP16 precision. At INT8 precision, v2 is closer to the default (FP16+v1) than INT8+v1, which implies that the enhancements made in v2 improve accuracy of the INT8 model.

FPS: AVs can tolerate lower camera FPS in easier scenarios as there is enough time to perceive the obstacle (even with the delay) and apply sufficient braking to avoid the collision. The results for the vehicle following scenario show that FPS of 3 is tolerable for the easier scenario but at least 5 FPS is required to remain safe for the harder scenario. Among the harder scenarios, cut-out and jaywalking require quicker response based on the scenario settings.

GPU frequency: For the baseline AV, the GPU processes 30 frames per second. This computation demand is satisfied by the default GPU settings as well as with 300MHz (the lowest setting we tested). The results can be different with a smaller GPU or when more perception algorithms are run per camera or with more cameras, testing which is part of our future work.

Based on these results, we find the following:

- High L1 norm of braking, throttle, and AV's y-coordinate do not always imply a safety concern. The planning algorithm changes how the AV's react to obstacles between the tested AV versions, which can affect the L1 norms but not the minimum distance. Simulation non-determinism (small change in many timesteps) can also make the L1 norms non-negligible. Minimum distance, however, may not change much even when the L1 norms change. So, we use the minimum distance as the primary metric.

- Small inaccuracies introduced by optimized models (e.g., INT8 versus FP16 model) show minimal effects on AV safety. Trading off small per-frame accuracy for significantly higher throughput will allow the designers to consider adding diversity from additional sensors to avoid possibility of consistent mispredictions and make the system safer.
- Camera FPS is the most safety sensitive parameter among the studied perception parameters, but shows significant tolerance even in the harder scenarios.
- Perception requirements for safe AV driving depend heavily on driving scenarios, i.e., the speed of AV, obstacles, and their distances. Based on these factors, the AV designers may predict the existence of some tolerance and consider optimizations to free up some resources for other tasks (e.g., diverse perception or a more complex planning algorithm).

Finding 1: AV system's tolerance to small per-frame inaccuracies and reduced FPS cameras even in the tested harder scenarios suggest the default system can tolerate even harder scenarios. It also suggests at the existence of opportunities for system optimization and introduction of more diversity to make the system safer.

4.6.4 Inaccurate World Model Prediction

To improve our understanding of the perception quality requirements, we introduce inaccuracies in the world model prediction using the models described in Section 4.5. Our results are shown in Figure 4.5, which uses the format similar to that of Figure 4.4 except that it does not show throttle L1 norm. We only show the results for the relatively harder scenarios here.

Positive and negative noise: Negative noise places the obstacle closer than its real location, which makes the AV act conservatively and brake harder (braking and y-coordinate L1 norms are higher). This phenomenon increases the minimum distance and results in no safety issues. For the positive noise, the results show that the AV can tolerate up to 50% for vehicle following and cut-in scenarios, 30% for jaywalking, and 10% for cut-out.

Random noise: Random noise introduces randomly generated positive or negative error per frame. Based on the frequency with which the negative noise is introduced, the planner can

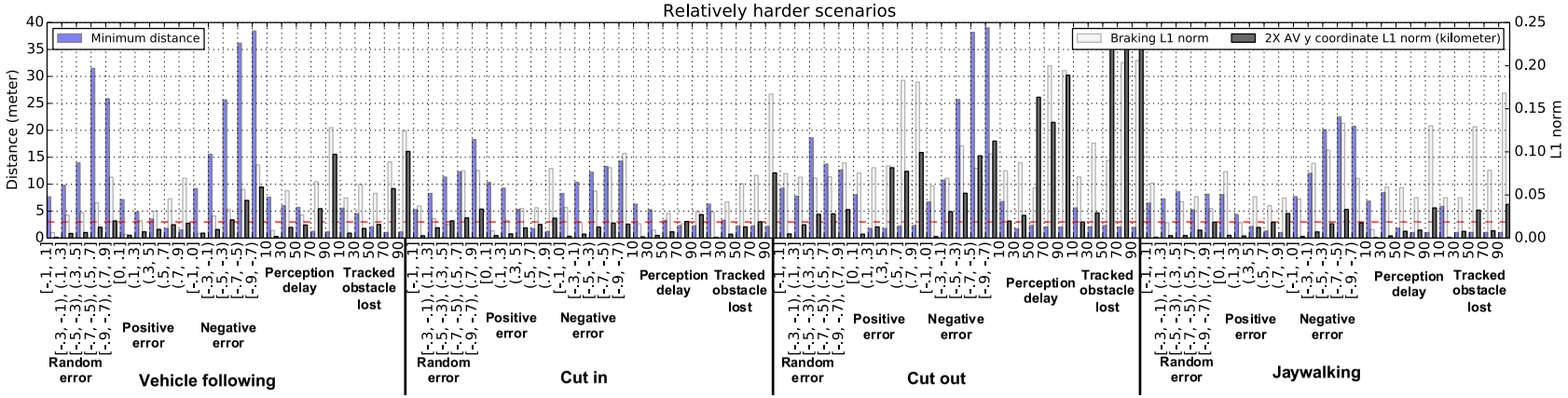


Figure 4.5. Effects on AV safety due to inaccuracies in the perceived world model.

take a conservative action by braking harder. While the results show that the minimum distance increases as the injected noise range increases (similar to negative noise), there are few exceptions in the jaywalking experiments. In these experiments, we hypothesis that the random noise introduces positive and negative error at different rates, which results in slight variations in the minimum distance trend. We do not observe any safety issues in the harder scenarios with this noise model.

Perception delay: The amount of tolerable delay depends heavily on the scenario. In the harder scenarios for vehicle following, cut-in, cut-out, and jaywalking, the AV tolerates significant delay of 50, 30, 10, and 30 frames, respectively, every 100 frames. We determine the tolerance based on the minimum distance metric (>3m is tolerable). For the vehicle following scenario, we interestingly find that 3 FPS is not tolerated, but a delay of 50 frames every 100 frames while running at 30 FPS is tolerated. We hypothesise that the AV was able to receive fast updated about the target's location after the delay, which allows the AV to react in time. If the speeds were higher, the tolerable delay would have been lower (as is the case in cut-out).

World model loss: This model results in more serious safety issues compared to the above models. The harder vehicle following, cut-in, cut-out, and jaywalking scenarios tolerate 30, 30, 10, and 10 frames of lost information, respectively, every 100 frames.

Based on these results, we find the following:

- As long as the planner is designed to take conservative actions based on the recent obstacle distance predictions, frequent negative noise introduced by the perception stack will not affect safety.
- While the tolerance to inaccurate world model depends heavily on the scenario, results from all the studied harder scenarios show tolerance to 10% positive noise or 10 frames of delay or 10 frames of the lost world model.
- The inaccurate world model results provide an efficient way to search the space to find optimal perception settings for safe operations. For example, the perception delay results showed low tolerance in the harder cut-out scenario, implying that the delay introduced by slower

processing units (e.g., limited GPU frequency) may not be tolerated.

Finding 2: The inaccurate world model prediction models provide an efficient way to test the tolerable perception limits to find optimal perception settings for a safe yet efficient AV. We observe that the tested AV has a tolerance to 10% noise and occasional 10 frames of delay or lost information in all the studied scenarios, which affirm Finding 1.

4.6.5 Sensitivity Analysis

The goal of this analysis is to quantify safety sensitivity of each of the design parameters. We use the data from Figures 4.4 and 4.5 to conduct the sensitivity study and show the results in Figure 4.6. We quantify the change in minimum distance metric value when a perception parameter's setting changes by presenting $10^{th} percentile(delta_p)$ in Figure 4.6(a). The delta values can be positive or negative. A negative value implies that the minimum distance has reduced (the system has become less safe) when a parameter is changed from a setting that is closer to default to a setting that is further (e.g., 30 FPS to 15 FPS for the camera). A positive value implies that the system has become safer. We show the 10^{th} percentile of maximum distance across all the scenarios (10^{th} $percentile(minimum\ distance_p)$) in Figure 4.6(b) for comparison. While we show $10^{th} percentile$ data here, similar graphs for *avg* and *min* can also be obtained. For both of the figures, the higher the value, the safer the AV. The setting range of a parameter significantly affects the sensitivity analysis. To better understand these effects, we analyze two parameter ranges – expanded and limited parameter ranges.

Expanded parameter ranges: We use these ranges – (1) camera FPS: 5, 6, 10, 15, 30; (2) GPU frequency(MHz): 300, 500, 800, 1000, 1200; (3) random, positive, and negative noise: up to 50% (or -50%); and (4) perception delay and world model loss: 10, 30, 50 frames. The delta values (left figure) and minimum distance values (right figure) suggest that the most safety sensitive parameter among the four SW/HW parameters is FPS. For inaccurate world model prediction, world model loss, perception delay, and positive noise worsen the safety compared to random noise and negative noise. These findings match the results shown in above two sections.

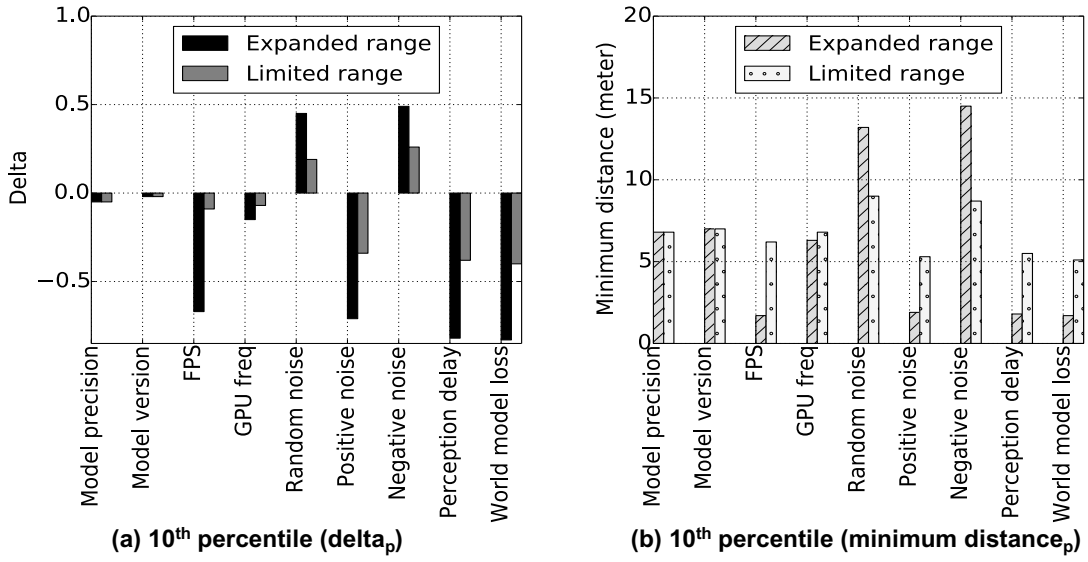


Figure 4.6. Sensitivity analysis.

These visualizations simplify the identification of the sensitive parameters, especially when many scenarios are evaluated.

Limited parameter ranges: We use these ranges – (1) camera FPS: 15, 30; (2) GPU frequency (MHz): 800, 1000, 1200; (3) random, positive, and negative noise: up to 10% (or -10%); and (4) perception delay and world model loss: 10 frames. Both delta and minimum distance-based results show no obvious safety issues among the hardware and software parameters, which implies that these parameters have similar negligible safety effects with the limited ranges. The inaccurate world model prediction results also show less safety sensitivity.

Finding 3: Sensitivity analysis presents AV designers with a simpler method to gain insights into the sensitive parameters for a quicker design space analysis.

4.6.6 Identifying Optimal but Safe Perception Settings

When designing an optimized AV system, the system designers may want to tune multiple perception parameters. While different strategies can be used to explore the space, we study a greedy approach. We prioritize the sensitive parameters and search by limiting the most-sensitive parameter setting. We demonstrate this approach by analyzing AV safety when the camera FPS,

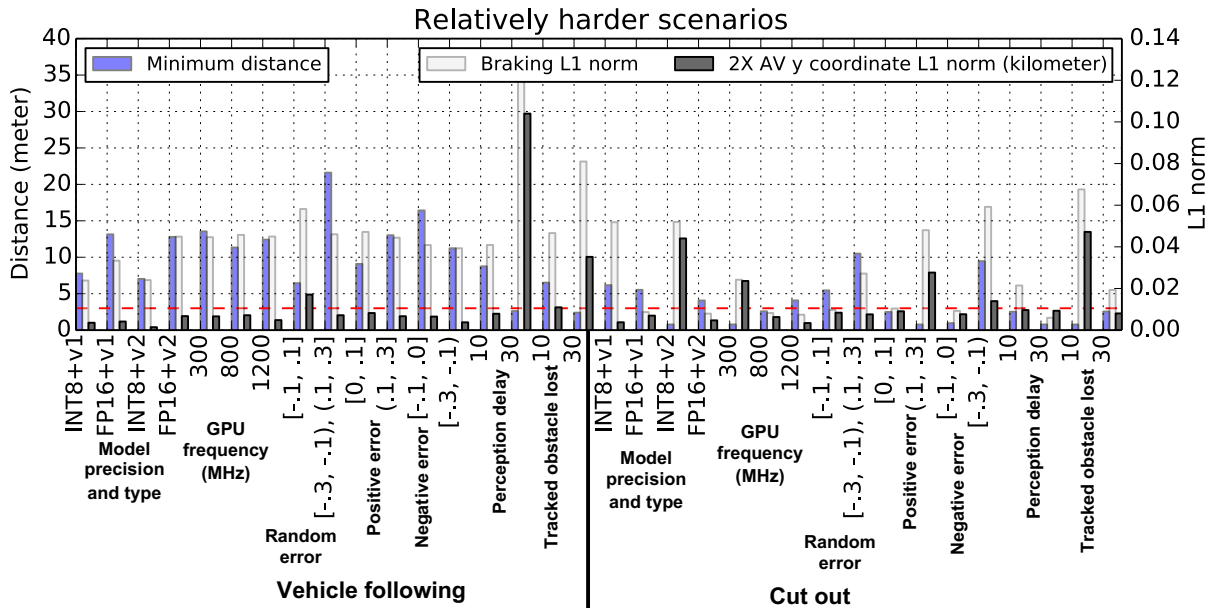


Figure 4.7. Fixed FPS analysis.

the most sensitive parameter, is set to a critical setting. A critical setting here refers to a value that makes the system unsafe when the value is reduced further. Due to space limitations, we show the results for two representative scenarios: vehicle following (critical FPS=5) and cut-out (critical FPS=10) in Figure 4.7. It uses the same format as that of Figure 4.5.

Model: INT8 model precision + model v2 results in a collision for the cut-out scenario (minimum distance is below the red dashed line). Unlike the results in Section 4.6.3, model precision + version combination has now become sensitive.

GPU frequency: Frequencies lower than 1200MHz result in collisions for the cut-out scenario, which demonstrates that the tolerance to higher latency inference has now diminished.

Noise: Similar to the findings in Section 4.6.4, negative noise does not introduce any safety concerns. Random error also does not introduce safety concerns, unless the noise moves the perceived object further from its real location between $t_2 - t_3$. Results show that the tolerance to positive error is reduced. For the vehicle following scenario, up to 30% positive noise is tolerated, but no positive noise in the cut-out scenario is tolerated.

Perception delay: We observe that up to 10 frames of perception delay is tolerated, but no delay

in the cut-out scenario is tolerated. This result corroborates with the intolerance observed when the GPU frequency was reduced.

World model loss: Similar to the above results, we observe 10 frames of the lost world model was tolerated in the vehicle following scenario, but no tolerance in the cut-out scenario.

Finding 4: Limiting a perception parameter to its critical setting (e.g., FPS=10 for cut-out) can reduce the resource demand (by 3× for cut-out), but removes tolerance in other perception parameters in the AV. Developing optimization strategies that tunes multiple design parameters to design an optimal and safer AV is a promising future direction.

4.6.7 Portability

Suraksha is a general AV safety evaluation framework. To demonstrate it's portability to a different software stack, we implemented it using LGSVL simulator [121] with Baidu's Apollo AV software stack [35]. The Apollo stack uses LiDAR as the primary perception module whereas the NVIDIA stack evaluated above uses camera as the primary sensor. We conducted limited experiments with the Apollo AV stack by varying LiDAR sensor parameters and injecting random noise into the world model before it is sent to the planner. While the detailed results were different (which is expected as the AV algorithms are different), the high-level safety tolerance trends are similar to the ones presented in Sections 4.6.3 and 4.6.4.

4.7 Related Work

Various AV simulation tools are available for AV developers, e.g., LGSVL simulator [121] and CARLA simulator [54]. These simulators can be connected to AV stacks such as Apollo [35] and Autoware [33]. Siemens' Simcenter [130] provides a leading physics-based simulation platform for robust testing of AV safety, reliability, and functionality. It allows users to test, develop, and optimize AV systems by following a systematic approach. This infrastructure is similar to NVIDIA DRIVE Constellation, which is a dedicated data center platform for AV hardware-in-the-loop simulation at scale[111]. It runs NVIDIA DRIVE Sim and enables AV

testing and validation efforts. Suraksha's concepts are general enough to be implemented on top of any of these simulators and AV stacks.

Safety models such as Responsibility-Sensitive Safety (RSS) [129] and Safety Force Field (SFF) [106] set rules for the AV to operate safely. These rules cross-check the AV's behavior with a separate model and keep safe space from potential hazards and avoid collisions. Some prior safety-aware AV system designs focus on developing better perception module algorithms to ensure AV safety [32, 60, 83]. Sina et al. [101] review safety challenges in machine learning techniques, and propose three methods to enhance safety of machine learning applications. Two studies explore safety assessment for connected AVs [114, 142, 153], and posit that safety can be improved if cars on the road can share information. Lin et al. [92] builds an autonomous driving system by integrating AV algorithms such as detection, tracking, and localization, and accelerate this system with four devices: GPUs, CPUs, FPGAs, and ASICs. This work performs latency analysis among four devices and studies the correlation between input image size and latency. However, these studies lack the capabilities to perform AV safety evaluation and comparison between different driving conditions. Suraksha is able to quantify and analyze AV safety and facilitate AV designers to optimize AV safety efficiently.

4.8 Conclusion

Current AV validation methodologies do not provide a mechanism to study sensitivity of AV components and their effects on safety. In this chapter, we propose an automated safety analysis framework, Suraksha, to explore an AV's component-level design choices using different driving scenarios generated based on a user-specified difficulty level. We employ Suraksha to analyze the safety implications of perception parameters that degrade the quality and present interesting insights that provide new information for AV designers to better optimize and design a safer AV.

Chapter 4 contains material from "Suraksha: A Framework to Analyze the Safety

Implications of Perception Design Choices in AVs", by Hengyu Zhao, Siva Kumar Sastry Hari, Timothy Tsai, Michael B. Sullivan, Stephen W. Keckler, Jishen Zhao, which appears in International Symposium on Software Reliability Engineering (ISSRE), Oct. 2021. The dissertation author was the primary investigator and author of this material.

Chapter 5

Conclusion

Safety is an essential component to be considered by AV developers when they design AV stacks. The safety issues exist in multiple aspects in a full stack AV system. In AV data centers, the performance bottleneck is caused by the significant data movement overhead in DNN training, which leads to energy inefficiency and long training time. In AV onboard systems, AV safety is highly sensitive to the onboard system end-to-end latency. Moreover, effective safety validation approaches are necessary to evaluate AV safety level before AVs are released to the public.

To solve the above mentioned problems, this dissertation presents three designs. Chapter 2 proposes to employ a processing-in-memory architecture in AV data centers to train DNN models efficiently, which addresses the data movement overhead in DNN training workloads. Chapter 3 presents the safety score, a latency based AV safety metric, and designs a safety-aware AV resource management scheme that optimize end-to-end latency. Chapter 4 proposes an AV safety validation framework, called Suraksha, and uses a case study to demonstrate Suraksha by analyzing safety effects of adjusting perception design choices.

Bibliography

[1] Audi and NVIDIA automotive partners. https://www.nvidia.com/en-us/self-driving-cars/partners/audi/.

[2] Intel Announces 250 Million Investment for Autonomous Driving. https://newsroom.intel.com/news/intel-announces-250-million-investment-autonomous-driving/#gs.3gdl7k.

[3] Ipmacc compiler. https://github.com/lashgar/ipmacc.

[4] Khronos Group, the open standard for parallel programming of heterogeneous systems. https://www.khronos.org/opencl/.

[5] Model S. https://www.tesla.com/models.

[6] NVIDIA CUDA. http://www.nvidia.com/cuda.

[7] Openarc. https://ft.ornl.gov/research/openarc.

[8] RISC-V: The free and open RISC instruction set architecture. https://riscv.org/.

[9] Self-driving cars: Understanding the 6 autonomous levels. https://www.fool.com/investing/2018/09/06/self-driving-cars-understanding-6-autonomous-level.aspx.

[10] Waymo. https://waymo.com/.

[11] Self-driving carts to make their debut on UC San Diego roads in January, 2017.

[12] Alphabet's Waymo is now in the 'prove it' stage of its self-driving journey, 2018.

[13] Baidu to integrate Mobileye's responsibility sensitive safety model into Apollo program, 2018.

[14] Illinois to develop testing program for self-driving vehicles, 2018.

[15] Texas A&M launches autonomous shuttle project in downtown Bryan, 2018.

[16] Uber advanced technologies group, 2019.

[17] Valeo signs an agreement with Mobileye to develop a new autonomous vehicle safety standard, 2019.

[18] NVIDIA, Profiler user's guide, http://docs.nvidia.com/cuda/profiler-users-guide/.

[19] Tensorflow, questions-words dataset, http://download.tensorflow.org/data/questions-words.txt.

[20] MultiModel: Multi-task machine learning across domains, https://ai.googleblog.com/2017/06/multimodel-multi-task-machine-learning.html.

[21] Intel, Vtune user's guide, https://software.intel.com/en-us/get-started-with-vtune/.

[22] NVIDIA, GeForce GTX 1080 Ti, https://www.nvidia.com/en-us/geforce/products/.

[23] NVIDIA, TITAN Xp, https://www.nvidia.com/en-us/geforce/products/10series/titan-xp/.

[24] TensorBoard: Visualizing learning, https://www.tensorflow.org/tensorboard.

[25] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[26] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 105–117, 2015.

[27] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, pages 336–348, 2015.

[28] Berkin Akin, Franz Franchetti, and James C. Hoe. Data reorganization in memory using 3D-stacked DRAM. In *Proceedings of the Annual International Symposium on Computer Architecture*, pages 131–143, 2015.

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

[30] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-layer CNN accelerators. In *MICRO*, pages 1–12. IEEE, 2016.

[31] Shaahin Angizi, Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. CMP-PIM: an energy-efficient comparator-based processing-in-memory neural network accelerator. In *Proceedings of the 55th Annual Design Automation Conference*, pages 105–110. ACM, 2018.

[32] Romuald Aufrère, Jay Gowdy, Christoph Mertz, Chuck Thorpe, Chieh-Chih Wang, and Teruko Yata. Perception for Collision Avoidance and Autonomous Driving. *Mechatronics*, 13(10):1149–1161, 2003.

[33] https://www.autoware.org/.

[34] Erfan Azarkhish, Davide Rossi, Igor Loi, and Luca Benini. Neurostream: Scalable and energy efficient deep learning with smart memory cubes. *IEEE Transactions on Parallel and Distributed Systems*, 29(2):420–434, 2018.

[35] Baidu Apollo Autonomous Driving Platform. https://apollo.auto/developer.html.

[36] Sabur Baidya, Yu-Jen Ku, Hengyu Zhao, Jishen Zhao, and Sujit Dey. Vehicular and Edge Computing for Emerging Connected and Autonomous Vehicle Applications. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[37] Subho S Banerjee, Saurabh Jha, James Cyriac, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on Over a Million Miles of Field Data. In *International Conference on Dependable Systems and Networks (DSN)*, pages 586–597, 2018.

[38] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[39] Anton Beloglazov and Rajkumar Buyya. Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing*, pages 826–831. IEEE Computer Society, 2010.

[40] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[41] Scott Boag, Parijat Dube, Benjamin Herta, Waldemar Hummer, Vatche Ishakian, Jayaram K. R., Michael Kalantar, Vinod Muthusamy, Priya Nagpurkar, and Florian Rosenberg. Scalable Multi-Framework Multi-Tenant Lifecycle Management of Deep Learning Training Jobs. In *Workshop on ML Systems at NIPS'17*, 2017.

[42] Teresa Brell, Ralf Philipsen, and Martina Ziefle. sCARy! Risk Perceptions in Autonomous Driving: The Influence of Experience on Perceived Benefits and Barriers. *Risk Analysis*,

39(2):342–357, 2019.

[43] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, 2017.

[44] Wolfram Burgard, Oliver Brock, and Cyrill Stachniss. Map-based precision vehicle localization in urban environments. In *MIT Press*, pages 352–359, 2011.

[45] Huangke Chen, Xiaomin Zhu, Hui Guo, Jianghan Zhu, Xiao Qin, and Jianhong Wu. Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment. *Journal of Systems and Software*, 99:20–35, 2015.

[46] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. DaDianNao: A machine-learning supercomputer. In *IEEE/ACM International Symposium on Microarchitecture*, 2014.

[47] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ISCA*, pages 367–379, 2016.

[48] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 27–39, 2016.

[49] Keonyup Chu, Minchae Lee, and Myoungho Sunwoo. Local path planning for off-road autonomous driving with avoidance of static obstacles. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1599–1616, 2012.

[50] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82. Springer, 1998.

[51] Yue Dang, Bin Wang, Ryan Brant, Zhiping Zhang, Maha Alqallaf, and Zhiqiang Wu. Anomaly detection for data streams in large-scale distributed heterogeneous computing environments. In *ICMLG2017 5th International Conference on Management Leadership and Governance*, page 121. Academic Conferences and publishing limited, 2017.

[52] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. Understanding and Optimizing Asynchronous Low-Precision Stochastic Gradient Descent. In *International Symposium on Computer Architecture (ISCA)*, 2017.

[53] Laurent Delobel, Claude Aynaud, Romuald Aufrere, Christophe Debain, Roland Chapuis,

Thierry Chateau, and Coralie Bernay-Angeletti. Robust localization using a top-down approach with several lidar sensors. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2371–2376. IEEE, 2015.

[54] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An Open Urban Driving Simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[55] Yasuko Eckert, Nuwan Jayasena, and Gabriel H. Loh. Thermal feasibility of die-stacked processing in memory. In *WoNDP*, pages 1–6, 2014.

[56] Hadi Esmaeilzadeh, Adrian Sampson, and Luis Ceze et al. Neural acceleration for general-purpose approximate programs. In *MICRO*, pages 449–460. IEEE Computer Society, 2012.

[57] OpenMP Forum. OpenMP Fortran application program interface, version 1.1. http://www.openmp.org, 1999.

[58] Daichi Fujiki, Scott Mahlke, and Reetuparna Das. In-memory data parallel processor. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 1–14, New York, NY, USA, 2018. ACM.

[59] Andrei Furda and Ljubo Vlacic. Towards Increased Road Safety: Real-time Decision Making for Driverless City Vehicles. In *International Conference on Systems, Man and Cybernetics (SMC)*, pages 2421–2426, 2009.

[60] Andrei Furda and Ljubo Vlacic. Enabling Safe Autonomous Driving in Real-world City Traffic Using Multiple Criteria Decision Making. *IEEE Intelligent Transportation Systems Magazine*, 3(1):4–17, 2011.

[61] Hans van Halteren, Jakub Zavrel, and Walter Daelemans. Improving accuracy in word class tagging through the combination of machine learning systems. *Computational linguistics*, 27(2):199–229, 2001.

[62] 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.

[63] 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.

[64] Jeff Hecht. Lidar for self-driving cars. *Optics and Photonics News*, 29(1):26–33, 2018.

[65] Georg Heigold, Vincent Vanhoucke, Alan Senior, Patrick Nguyen, Marc'Aurelio Ranzato, Matthieu Devin, and Jeffrey Dean. Multilingual acoustic models using distributed deep neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8619–8623. IEEE, 2013.

[66] Tove Helldin, Göran Falkman, Maria Riveiro, and Staffan Davidsson. Presenting system uncertainty in automotive uis for supporting trust calibration in autonomous driving. In *Proceedings of the 5th international conference on automotive user interfaces and interactive vehicular applications*, pages 210–217. ACM, 2013.

[67] Michael Himmelsbach, Andre Mueller, Thorsten Lüttel, and Hans-Joachim Wünsche. Lidar-based 3D object perception. In *Proceedings of 1st international workshop on cognition for technical systems*, volume 1, 2008.

[68] HMCC. Hybrid memory cube specification 2.0. http://http://www.hybridmemorycube.org/.

[69] International Organization for Standardization. "26262: Road vehicles-functional safety", 2011.

[70] AV Safety Ventures Beyond ISO 26262. https://www.eetimes.com/document.asp?doc_id=1334397#.

[71] ISO/PAS 21448. https://www.iso.org/standard/70939.html.

[72] Animesh Jain, Amar Phanishayee, Jason Mars, Lingjia Tang, and Gennady Pekhimenko. Gist: Efficient data encoding for deep neural network training. In *ISCA*, pages 1–14, 2018.

[73] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[74] Kichun Jo, Junsoo Kim, Dongchul Kim, Chulhoon Jang, and Myoungho Sunwoo. Development of autonomous car part II: A case study on the implementation of an autonomous driving system based on distributed architecture. *IEEE Transactions on Industrial Electronics*, 62(8):5119–5132, 2015.

[75] Avdhut Joshi and Michael R James. Generation of accurate lane-level maps from coarse prior maps and lidar. *IEEE Intelligent Transportation Systems Magazine*, 7(1):19–29, 2015.

[76] Nidhi Kalra and Susan M Paddock. Driving to Safety: How Many Miles of Driving Would it Take to Demonstrate Autonomous Vehicle Reliability? *Transportation Research Part A: Policy and Practice*, 94:182–193, 2016.

[77] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 380–392, 2016.

[78] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. Selecta: Heterogeneous cloud storage configuration for data analytics. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, pages 759–773, 2018.

[79] Philip Koopman and Frank Fratrik. How many operational design domains, objects, and events? *SafeAI@ AAAI*, 4, 2019.

[80] Philip Koopman and Michael Wagner. Autonomous Vehicle Safety: An Interdisciplinary Challenge. *IEEE Intelligent Transportation Systems Magazine*, 9(1):90–96, 2017.

[81] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, 2002.

[82] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[83] Felix Kunz, Dominik Nuss, Jürgen Wiest, Hendrik Deusch, Stephan Reuter, Franz Gritschneder, Alexander Scheel, Manuel Stübler, Martin Bach, Patrick Hatzelmann, Cornelius Wild, and Klaus Dietmayer. Autonomous Driving at Ulm University: A Modular, Robust, and Sensor-independent Fusion Approach. In *Intelligent Vehicles Symposium (IV)*, pages 666–673, 2015.

[84] Christian Laugier, Igor Paromtchik, Mathias Perrollaz, Yong Mao, John-David Yoder, Christopher Tay, Kamel Mekhnacha, and Amaury Nègre. Probabilistic analysis of dynamic scenes and collision risk assessment to improve driving safety. *Its Journal*, 3(4):4–19, 2011.

[85] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[86] Janghaeng Lee, Mehrzad Samadi, Yongjun Park, and Scott Mahlke. Transparent CPU-GPU collaboration for data-parallel kernels on heterogeneous systems. In *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques*, pages 245–256, 2013.

[87] John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Sertac Karaman, Olivier Koch, Yoshiaki Kuwata, David Moore, Edwin Olson, Steve Peters, Justin Teo, Robert Truax, Matthew

Walter, David Barrett, Alexander Epstein, Keoni Maheloni, Katy Moyer, Troy Jones, Ryan Buckley, Matthew Antone, Robert Galejs, Siddhartha Krishnamurthy, and Jonathan Williams. A Perception-driven Autonomous Urban Vehicle. *Journal of Field Robotics*, 25(10):727–774, 2008.

[88] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, Michael Sokolsky, Ganymed Stanek, David Stavens, Alex Teichman, Moritz Werling, and Sebastian Thrun. Towards Fully Autonomous Driving: Systems and Algorithms. In *Intelligent Vehicles Symposium (IV)*, pages 163–168, 2011.

[89] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In *International Symposium on Software Reliability Engineering (ISSRE)*, pages 25–36, 2020.

[90] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–480, 2009.

[91] Tian Li, Jie Zhong, Ji Liu, Wentao Wu, and Ce Zhang. Ease.ml: Towards multi-tenant resource sharing for machine learning workloads. *Proc. VLDB Endow.*, 11(5), 2018.

[92] 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 *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 751–766, 2018.

[93] Shaoshan Liu, Liyun Li, Jie Tang, Shuang Wu, and Jean-Luc Gaudiot. Creating Autonomous Vehicle Systems. *Synthesis Lectures on Computer Science*, 6(1):i–186, 2017.

[94] Gabriel H. Loh, Nuwan Jayasena, Mark H. Oskin, Mark Nutter, David Roberts, Mitesh Meswani, Dongping Zhang, and Mike Ignatowski. A processing-in-memory taxonomy and a case for studying fixed-function PIM. In *WoNDP*, pages 1–6, 2013.

[95] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 190–200, New York, NY, USA, 2005.

[96] Muthucumaru Maheswaran, Shoukat Ali, HJ Siegal, Debra Hensgen, and Richard F Freund. Dynamic matching and scheduling of a class of independent tasks onto hetero-

geneous computing systems. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pages 30–44. IEEE, 1999.

[97] Bernard Marr. Key milestones of Waymo – Google's self-driving cars. https://www.forbes.com/sites/bernardmarr/2018/09/21/key-milestones-of-waymo-googles-self-driving-cars/.

[98] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[99] Vicente Milanés, Steven E Shladover, John Spring, Christopher Nowakowski, Hiroshi Kawazoe, and Masahide Nakamura. Cooperative Adaptive Cruise Control in Real Traffic Situations. *IEEE Transactions on Intelligent Transportation Systems*, 15(1):296–305, 2013.

[100] Sparsh Mittal and Jeffrey S. Vetter. A survey of CPU-GPU heterogeneous computing techniques. *ACM Comput. Surv.*, 47(4):69:1–69:35, July 2015.

[101] Sina Mohseni, Mandar Pitale, Vasu Singh, and Zhangyang Wang. Practical Solutions for Machine Learning Safety in Autonomous Vehicles. *arXiv preprint arXiv:1912.09630*, 2019.

[102] Joanna Moody, Nathaniel Bailey, and Jinhua Zhao. Public Perceptions of Autonomous Vehicle Safety: An International Comparison. *Safety Science*, 121:634–650, 2020.

[103] Lifeng Nai and Hyesoon Kim. Instruction offloading with HMC 2.0 standard: A case study for graph traversals. In *Proceedings of the 2015 International Symposium on Memory Systems*, pages 258–261, 2015.

[104] NCAP Roadmap 2016-2020. https://www.euroncap.com/en/about-euro-ncap/timeline/roadmap-2016-2020.

[105] NHTSA, Automated Driving Systems: A Vision for Safety, 2017. https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/13069a-ads2.0_090617_v9a_tag.pdf.

[106] David Nistér, Hon-Leung Lee, Julia Ng, and Yizhou Wang. The Safety Force Field. NVIDIA White Paper, 2019.

[107] NVIDIA. cudnn. https://developer.nvidia.com/cudnn.

[108] GeForce RTX 2080 Graphics Card — NVIDIA, 2020.

[109] The World's Most Powerful Graphics Card — NVIDIA TITAN V, 2020.

[110] NVIDIA DRIVE - Software — NVIDIA Developer, 2021.

[111] NVIDIA DRIVE Constellation — NVIDIA Developer, 2021.

[112] NVIDIA DRIVE Sim — NVIDIA Developer, 2021.

[113] Umit Ozguner, Christoph Stiller, and Keith Redmill. Systems for Safety and Autonomous Behavior in Cars: The DARPA Grand Challenge Experience. *Proceedings of the IEEE*, 95(2):397–412, 2007.

[114] Alkis Papadoulis, Mohammed Quddus, and Marianna Imprialou. Evaluating the Safety Impact of Connected and Autonomous Vehicles on Motorways. *Accident Analysis & Prevention*, 124:12–22, 2019.

[115] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[116] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 267–278, 2016.

[117] James Reinders. Vtune performance analyzer essentials. *Intel Press*, 2005.

[118] Andreas Reschka. Safety Concept for Autonomous Vehicles. In *Autonomous Driving*, pages 473–496. Springer, 2016.

[119] Minsoo Rhu, Natalia Gimelshein, and Jason Clemons et al. vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *MICRO*, 2016.

[120] Minsoo Rhu, Mike O'Connor, Niladrish Chatterjee, Jeff Pool, Youngeun Kwon, and Stephen W Keckler. Compressing DMA engine: Leveraging activation sparsity for training deep neural networks. In *HPCA*, pages 78–91, 2018.

[121] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving. *arXiv preprint arXiv:2005.03778*, 2020.

[122] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[123] SAE J3016 automated-driving graphic. https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic.

[124] SAE. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles J3016_201806. https://www.sae.org/standards/content/j3016_201806/.

[125] Sebastian Schneider, Michael Himmelsbach, Thorsten Luettel, and Hans-Joachim Wuensche. Fusing vision and lidar-synchronization, correction and occlusion reasoning. In *2010 IEEE Intelligent Vehicles Symposium*, pages 388–393. IEEE, 2010.

[126] Fabian Schuiki, Michael Schaffner, Frank K. Gürkaynak, and Luca Benini. A scalable near-memory architecture for training deep neural networks on large in-memory datasets. *arXiv*, abs/1803.04783, 2018.

[127] Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: a local SVM approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 32–36. IEEE, 2004.

[128] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 14–26, 2016.

[129] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a Formal Model of Safe and Scalable Self-driving Cars. *arXiv preprint arXiv:1708.06374*, 2017.

[130] Simcenter: ADAS & AV System Simulation. https://www.plm.automation.siemens.com/global/en/products/simulation-test/active-safety-system-simulation.html.

[131] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[132] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: modeling and implementation. *ACM Transactions on Architecture and Code Optimization*, 1(1):94–125, 2004.

[133] Jingwei Song, Jun Wang, Liang Zhao, Shoudong Huang, and Gamini Dissanayake. MIS-SLAM: Real-time large scale dense deformable slam system in minimal invasive surgery based on heterogeneous computing. *arXiv preprint arXiv:1803.02009*, 2018.

[134] Gilbert Strang. *Introduction to Linear Algebra*, volume 3. Wellesley-Cambridge Press Wellesley, MA, 1993.

[135] Synopsys. Design compiler. https://www.synopsys.com/support/training/rtl-synthesis/design-compiler-rtl-synthesis.html.

[136] Synopsys. Primetime. https://www.synopsys.com/support/training/signoff/primetime1-fcd.html.

[137] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[138] Michael Tuchler, Andrew C Singer, and Ralf Koetter. Minimum mean squared error equalization using a priori information. *IEEE Transactions on Signal processing*, 50(3):673–683, 2002.

[139] Jessica Van Brummelen, Marie O'Brien, Dominique Gruyer, and Homayoun Najjaran. Autonomous Vehicle Perception: The Technology of Today and Tomorrow. *Transportation Research Part C: Emerging Technologies*, 89:384–406, 2018.

[140] Jaycil Z Varghese and Randy G Boone. Overview of Autonomous Vehicle Sensors and Systems. In *International Conference on Operations Excellence and Service Engineering*, pages 178–191, 2015.

[141] Peiqi Wang, Yu Ji, Chi Hong, Yongqiang Lyu, Dongsheng Wang, and Yuan Xie. SNrram: an efficient sparse neural network computation architecture based on resistive random-access memory. In *Proceedings of the 55th Annual Design Automation Conference*, page 106. ACM, 2018.

[142] Lanhang Ye and Toshiyuki Yamamoto. Evaluating the Impact of Connected and Autonomous Vehicles on Traffic Safety. *Physica A: Statistical Mechanics and its Applications*, 526:121009, 2019.

[143] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism. In *International Symposium on Computer Architecture (ISCA)*, 2017.

[144] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *arXiv preprint arXiv:1906.05113*, 2019.

[145] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[146] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L. Greathouse, Lifan Xu, and Michael Ignatowski. TOP-PIM: Throughput-oriented programmable processing

in memory. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, pages 85–98, 2014.

[147] Hengyu Zhao, Jiawen Liu, Matheus A. Ogleari, Dong Li, and Jishen Zhao. Processing-in-memory for energy-efficient neural network training: A heterogeneous approach. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 1–14, 2018.

[148] Hengyu Zhao, Yubo Zhang, Pingfan Meng, Hui Shi, Li Erran Li, Tiancheng Lou, and Jishen Zhao. Towards Safety-Aware Computing System Design in Autonomous Vehicles. *arXiv preprint arXiv:1905.08453*, 2019.

[149] Hengyu Zhao, Yubo Zhang, Pingfan Meng, Hui Shi, Li Erran Li, Tiancheng Lou, and Jishen Zhao. Driving scenario perception-aware computing system design in autonomous vehicles. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 88–95. IEEE, 2020.

[150] Hengyu Zhao, Yubo Zhang, Pingfan Meng, Hui Shi, Li Erran Li, Tiancheng Lou, and Jishen Zhao. Safety score: A quantitative approach to guiding safety-aware autonomous vehicle computing system design. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1479–1485. IEEE, 2020.

[151] Bowen Zheng, Yue Gao, Qi Zhu, and Sandeep Gupta. Analysis and optimization of soft error tolerance strategies for real-time systems. In *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*, pages 55–64. IEEE Press, 2015.

[152] Bowen Zheng, Hengyi Liang, Qi Zhu, Huafeng Yu, and Chung-Wei Lin. Next generation automotive architecture modeling and exploration for autonomous driving. In *2016 IEEE computer society annual symposium on VLSI (ISVLSI)*, pages 53–58. IEEE, 2016.

[153] Bowen Zheng, Chung-Wei Lin, Huafeng Yu, Hengyi Liang, and Qi Zhu. Convince: A cross-layer modeling, exploration and validation framework for next-generation connected vehicles. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2016.

[154] Qi Zhu, Haibo Zeng, Wei Zheng, Marco DI Natale, and Alberto Sangiovanni-Vincentelli. Optimization of task allocation and priority assignment in hard real-time distributed systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(4):85, 2012.

[155] Yuxiong Zhu, Borui Wang, Dong Li, and Jishen Zhao. Integrated thermal analysis for processing in die-stacking memory. In *Proceedings of the Second International Symposium on Memory Systems*, pages 402–414, 2016.